IMPLEMENTATION OF A THEOREM-PROVER BASED ON WFF-RESOLUTION WITH EQUALITY

A Thesis Submitted
in Partial Fullfilment of the Requirements
for the degree of

MASTER OF TECHNOLOGY

By

MANOHAR LAL

to the

Department of Computer Science and Engineering
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

MAY, 1991.

ABSTRACT

In this thesis, a theorem-prover based on *WFF-Resolution with equality* is implemented. The prover is interactive. Due to some conditions leading to incompleteness in the original *WFF-Resolution with equality* rule, the rule for factoring has been modified. Also, a modified definition of *dependency of an existential variable* is suggested, which is simpler than the earlier one and is efficient computationally. Using this definition unification sets will be smaller.

ABSTRACT

In this thesis, a theorem-prover based on *WFF-Resolution with equality* is implemented. The prover is interactive. Due to some conditions leading to incompleteness in the original *WFF-Resolution with equality* rule, the rule for factoring has been modified. Also, a modified definition of *dependency of an existential variable* is suggested, which is simpler than the earlier one and is efficient computationally. Using this definition unification sets will be smaller.

ACKNOWLEDGEMENTS

Thanks giving o thou,
who has given us so much,
mercifully grant us one
thing more -- a grateful heart.
George Herbert

I have very much appreciated having Dr. H. Karnick as my supervisor. He has been for me a great source of motivation and encouragement. His ever-smiling nature coupled with his having complete understanding of the problems one faces in everyday life, and his tolerance to the shortcomings of others make him a most likeable person, who makes you feel comforfortable right away. I am highly indebted to him for his able guidance and for many useful suggestions that improved the quality of this thesis to a large extent.

Also, I had the opportunity of meeting and having discussion with many other talented people. P.V. Ravishanker is one of those rare people who is able to grasp the true nature of a problem almost immediately. He saved me many an hour (rather day) in locating and correcting some very subtle errors in my LISP code. Also, Anup Chawla, Deepak Gupta, C. R. S. Reddy, K.S. Venktesh (EE), Deepak Murthy (EE) provided very useful help and suggestions in their respective areas. I had the opportunity of fruitful discussions with Kalyan Basu, Pawan Kumar, Brijesh Pandey, B. Srinivas, Sreesh Jadav, V. Ramesh and many others. All of these are highly capable people, who were always ready to help. I express my heart-felt gratitude to all of them.

I express my sincere thanks to the faculty in the CSE department for their helpful attitude and kind attention they paid to me. Also, I thank them all for extending my fellowship for a period of five months.

I express my gratitude to Software engineers Mr. B. M. Shukla and Brahmaji Rao, to Mr. Bajpai, Panditji, Mr. Tiwari and Mr. Shukla of the Lab. staff, to the office staff, the departmental library staff and the central library staff for their invaluable help to me.

Prof. Tulsi Ram, Prof. B. D. Sharma, Prof.G. C. Ahuja, Dr. S. S. Rana, and Dr. B. S. Sidhu, all have been a great source of inspiration, encouragement and help

throughout my career. I am sincerely thankful to all of them.

I am very much indebted to Pt. Deep Narayan Tiwari of I. I. T. Gate, Kanpur for his guidance and help in spiritual matters, which proved very useful during periods of distress and adversity.

I am thankful to many of my hostel-mates for their pleasant company and for providing home-like environment.

I express my gratitude to my dear friend Vasudevan of Prentice-Hall of India for supporting me with the books I need from time to time.

Last, but not the least, I come to the people who have suffered so much due to my *"maniac academicia"*-- my mother, my wife Santosh, and my daughter Meenakshi. I realized infinity through their patience. No words of thanks and gratitude can compensate for the sacrifice they have to make for my madness. Still, I say *'thank you all'*. Also, I thank Lucy, Juli and Chameli (at Kanpur), the furry members of our family, for their affection and for relieving me of fatigue many a time through their playful acts.

MANOHAR LAL

CONTENTS

Chapter 1

INTRODUCTION

The beauty of a theorem from mathematics,
the preciseness of an inference rule of logic,
the intrigue of a puzzle, and
the challenge of a game—
all are present in the field of
automated reasoning

Larry Wos

1.1 REASONING AND ITS AUTOMATION

Problem-solving is an essential and dominant part of human activity. A *problem* exists whenever objects must satisfy certain conditions given some set of constraints or facts. A *solution* is a way of satisfying the conditions. Any solution to a problem involves some form of *reasoning*. While we can reason by analogy or inductively, conclusions do not *necessarily* follow from the given facts or constraints. Deductive reasoning provides this sort of power. In *deductive reasoning*, a set of inference rules are used to obtain statements from a given set of facts. Using sound inference rules, new statements derived *inevitably* follow from the given facts. Intuitively, an inference rule is *sound* if any conclusion derived by using it is true whenever the premisses (i.e., the given facts) are true. Alternatively, a rule is *unsound* when a model can be found in which the premisses hold, but some conclusion derived from the rule does not.

1.2 BRIEF OVERVIEW OF THEOREM PROVING USING RESOLUTION

Basically, there are two different approaches for proving a theorem. One

of them is to start with the set of axioms and arrive at a conclusion. The other is the well-known *reductio ad absurdum* in which we start with a negation of the conclusion and derive a contradiction. The latter approach more commonly known as a *refutation procedure* is one on which many of the theorem provers, specially those developed during late seventies (1965–70) are based.

The explosion of interest in the field of automated theorem-proving can be traced to Robinson's method of resolution [Rob 1965], which is a refutation procedure. Most of the theorem-provers in the tradition of the resolution method, first convert *well-formed-formula* (*WFF*) or sentences of FOPC (*First-Order Predicate Calculus*) axioms and the negated theorem, to a set of clauses. A *clause* is a disjunction of literals, where a *literal* is a positive or negative atom.

1.2.1 Resolution Principle

The resolution principle is a single rule of inference that employs a substitution device called unification. *Unification* is the process of finding a substitution that makes two terms identical. The inference rule is complete for FOPC. The rule may be stated as follows:

If $A \vee L_1$ and $B \vee \sim L_2$ are variable-disjoint clauses (A and B are clauses; L_1 and L_2 are literals), then from $A \vee L_1$ and $B \vee \sim L_2$ (the *parent* clauses) deduce clause $A\theta \vee B\theta$ (the *resolvent*) where θ is a substitution such that $L_1\theta$ and $L_2\theta$ are identical. The substitution θ is special in that it is the most general substitution that makes $L_1\theta$ and $L_2\theta$ equal. Thus any substitution that would make these two

literals equal, is an instance of θ. This is a powerful tool because it eliminates branching of search due to different possible substitutions that make the two atoms identical but lead to different clauses.

One uses the resolution inference rule to generate clauses until for some literal L, literals L and ~L are generated. Since their conjunction is a contradiction, the original formula is unsatisfiable.

To illustrate the application of the rule, we consider below an example.

Premises:

(1) The custom officials searched everyone who entered this country who was not a VIP.

(2) Some of the drug pushers entered this country and they were only searched by drug pushers.

(3) No drug pusher was a VIP.

Conclusion:

Some of the officials were drug pushers.

Let the predicate symbols be defined as below:

$E(x)$: *x entered the country,*

$V(x)$: *x was a VIP,*

$S(x,y)$: *y searched x,*

$C(x)$: *x was a custom official,*

$P(x)$: *x was a drug pusher.*

The FOL (*First Order Logic*) translation of the above problem, with the negated conclusion, consists of

S_1: $(\forall x)\,[(E(x) \wedge \sim V(x)) \rightarrow (\exists y)\,(S(x, y) \wedge C(y))]$

S_2: $(\exists x)\,[P(x) \wedge E(x) \wedge (\forall y)\,(S(x, y) \rightarrow P(y))]$

S_3: $(\forall x)\,[P(x) \rightarrow \sim V(x)]$

S_4: $\sim(\exists x)\,[P(x) \wedge C(x)]$.

Transforming these formulae into clauses, we obtain

(1) $\sim E(x) \vee V(x) \vee S(x, f(x))$

(2) $\sim E(x) \vee V(x) \vee C(f(x))$

(3) $P(a)$

(4) $E(a)$

(5) $\sim S(a,y) \vee P(y)$

(6) $\sim P(x) \vee \sim V(x)$

(7) $\sim P(x) \vee \sim C(x)$.

The resolution proof is as follows:

(8)	$\sim V(a)$	from (3) and (6)
(9)	$V(a) \vee C(f(a))$	from (2) and (4)
(10)	$C(f(a))$	from (8) and (9)
(11)	$V(a) \vee S(a, f(a))$	from (1) and (4)
(12)	$S(a, f(a))$	from (8) and (11)
(13)	$P(f(a))$	from (12) and (5)
(14)	$\sim C(f(a))$	from (13) and (7)
(15)	null	from (10) and (14).

For detailed consideration of the method with illustrations, refer to [ChL 73], [Lov 78], [Bha 88] and [Wos 83].

1.3 SOME EXTENSIONS OF RESOLUTION PRINCIPLE

One drawback with resolution theorem proving is its use of clausal form. Reasoning about wffs in terms of clauses is somewhat like doing arithmatic of integers in terms of sequences of ones representing integers. Due to conversion pragmatically useful information encoded in the choice of logical connectives and quantifiers, is lost. For example, the sense of a chaining approach to deduction in the connective $\rightarrow$ (conditional) is replaced by the sense of case analysis embedded in the connective $\vee$ (disjunction). Also a sentence may break into a large number of clauses, resulting in substantial redundancy in the resolution search space. Further, the clausal form is difficult to read and is not human-oriented. Guiding the theorem prover becomes difficult. This is a serious drawback in view of the fact that generating proofs of interesting, non-trivial theorems requires human interaction.

Murray [Mur 82] proposed Non-Clausal resolution (NC-resolution) which requires removal of quantifiers from formulae but keeps the logical connectives intact. He also established the completeness of the method. NC-resolution reduces to binary resolution in the case of clauses.

A further step would be when we can resolve on formulae as they are with quantifiers in place. A resolution rule called *WFF-resolution* has been introduced by Bhatta [Bha 88], by which we can resolve on formulae as they are. The inference rule is sound as shown in Bhatta [Bha 88].

The next chapter includes detailed discussion of WFF-resolution.

1.4 EQUALITY IN REASONING

The axioms of equality play a crucial role in automated reasoning, because of their almost universal application to axioms of particular theories. However, even for a theory with a small number of axioms, the number of equalities may become very large. *Paramodulation* [Wor 69] and *E-resolution* [Mor 69] were serious attempts at dispensing with explicit use of these axioms in refutations and yet be able to reason about equality. But paramodulation is intractable and E-resolution is incomplete.

A successful attempt in this direction, has been made by Digricoli and Harrison [DiH 86]. The two rules of inference , *RUE* (Resolution by Unification and Equality) and *NRF* (Negative Reflective Function) have been introduced in [DiH 86] that are primarily based on the resolution principle and are discussed only for the clausal case.

The rules of inference can be stated as follows:

RUE rule of inference

The RUE resolvent of two clauses $A \vee P(s_1,s_2,\ldots,sn)$ and $B \vee \sim P(t_1,t_2,\ldots,tn)$ (with A and B as clauses and P a particular predicate symbol), is $A\theta \vee B\theta \vee D$, where θ is a substitution and D is a disjunction of inequalities specified by a disagreement set of $P(s_1,s_2,\ldots,sn)\theta$ and $P(t_1,t_2,\ldots,tn)\theta$.

NRF rule of inference

The NRF resolvent of $A \vee t_1 \neq t_2$ (A is a clause) is $A\theta \vee D$, where θ is a substitution and D is a disjunction of inequalities specified by a disagreement

set of $t_1\theta$ and $t_2\theta$.

A *disagreement set* of two complementary literals or of the two terms of a negative equality, is a set of pairs of non-identical terms at corresponding argument places.

The equality-based resolution has been extended by Bhatta [Bha 88] to NC-resolution as well as to WFF-resolution, calling the corresponding inference rules as *NCRUE* (alongwith *NCNRF*) and *WRUE* (alongwith *WNRF*) . The detailed discussion of WRUE and WNRF is given in the next chapter.

In the remaining sections of this chapter, we talk about some aspects of the thesis.

1.5 SCOPE AND OBJECTIVES

This project implements the inference rule WFF-resolution with equality [Bha 88], which accepts and operates on closed wffs of FOPC in their original forms.

It is a user-friendly interactive system giving appropriate messages whenever desirable. It has an in-built type-checking mechanism. In case user supplies a value of inappropriate type, the system instead of exiting , goes on prompting the user until a value of appropriate type is supplied.

The system allows the user to supply formulae that are more human-oriented including restricted use of natural language elements. It incorporates mechanisms vert human-oriented forms of representation of formulae to forms suitable nipulation by machine and vice-versa.

1.6 THE LANGUAGE AND THE SYSTEM

The theorem-prover has been implemented in *Domain Common LISP* on the *Appolo.DN 3000 version AEGIS_DOMAIN/IX*.

1.7 OVERVIEW OF THE THESIS

An overview of the full thesis is as follows:

Chapter 2 describes the method of WFF-resolution with equality.

Chapter 3 implements this method, so that for a given pair of formulae, its resolvent/resolvents is/are returned automatically. Further for a given formula, its wnrf factor/factors is/are also generated again automatically.

In Chapter 4, we give concluding remarks and also mention about various possibilities for extension of the theorem-prover.

Appendix 1 includes hints for efficient use of the theorem-prover and solution of some problems.

Appendix 2 contains the (LISP) code of the theorem-prover.

Chapter 2

WFF-RESOLUTION WITH EQUALITY: THE METHOD

Method consists entirely in properly ordering and arranging the things to which we should pay attention

Descartes: *Oeuvres*, vol. X, p. 379,
Rules for the direction of mind, Rule V.

A Method of solution is perfect if we can forsee from the start, and even prove, that following that method we shall attain our aim

Leibnitz: *Opuscules*, p. 161.

We are already aware of the disadvantages of the automated reasoning systems that compulsorily require clausal form of representation or the ones that have no built-in mechanism for handling equalities without explicit use of equality axioms.

We mentioned that Murray's [Mur 82] NC-resolution operates on formulae containing all connectives but with quantifiers removed. However, removal of quantifiers creates some inherent problems for NC-resolution systems. The two major problems are

(a) loss of intuition behind selecting appropriate quantifiers and

(b) sentences becoming too complex, when the quantifiers within the scope of

equivalences are removed.

In the next section we discuss *WFF-resolution* which is NC-resolution with quantifiers in place. Then we extend WFF-resolution by incorporating equality.

We assume that the reader is familiar with the notations and concepts introduced in [Mur 82], [DiH 86] and [Bha 88]. We only define some of the key concepts required in the statements of the new inference rules. The definitions are on an informal level. For more rigorous definitions and illustrations one should refer to [Bha 88].

2.1 WFF-RESOLUTION

Under this scheme, the information provided by Skolem functions in clausal resolution, is captured within the terms of a wff. In this context, the definitions of a term and related concepts will be slightly different from the ones that are used in the context of a well-formed formula, clausal resolution and NC-resolution.

In order to find appropriate subwffs of a pair of WRUE-resolvable formulae, on which resolution can be performed, we need the

2.1.1 Definition (*Propositional Structure of a Wff*)

Propositional structure of a formula W of FOPC is a formula of propositional calculus obtained by dropping all the quantifiers from the formula and by replacing each occurrence of an atom by its predicate-symbol.

For example, the propositional structure of the formula

$$(\forall x)[\, P(x, a) \vee (\forall y)\, (\sim Q(x,y) \rightarrow R(y))]$$

of FOPC is the formula

$$P \vee (\sim Q \rightarrow R),$$

of propositional calculus.

Next, we define the polarity of a subwff in a formula.

2.1.2 Definition *(Polarity of a Subwff in a Wff)*

Let F, S_1 and S_2 be formulae and F be a subwff of S_1.

(1) It is positive for subwff F in F.

(2) If F is positive (negative) in S_1, then it is negative (positive) in $\sim S_1$ and $(S_1 \rightarrow S_2)$.

(3) If F is positive (negative) in S_1, then it is positive (negative) in $(S_1 \wedge S_2)$, $(S_1 \vee S_2)$ and $(S_2 \rightarrow S_1)$.

(4) It is both positive and negative in $(S_1 \leftrightarrow S_2)$.

For example, in formula

W: $\sim[\, P(x) \rightarrow [\, (\, Q(x) \leftrightarrow R(x)\,) \rightarrow T(x)\,]\,]$,

both $Q(x)$ and $R(x)$ are bipolar (have both negative and positive polarities), whereas $T(x)$ is of negative polarity and $P(x)$ is of positive polarity in W.

For resolution of two formulae first we determine the propositional structure say *struc* of the subwffs on which resolution may take place and also the polarity say *pol* of the subwffs in the first formula. Then we take a non-empty subset* of the set of all subwffs of the first formula, which have propositional structure *struc* and which occur with polarity *pol* in the first formula, similiarly we take a non-empty subset of the set of all the subwffs of the second formula which have propositional structure *struc* and occur with polarity other than *pol* in the second formula. Next, all the subwffs in these two subsets are unified. The

unification takes place in terms of sequences of quantified terms of the subwffs, where the concept of sequence of quantified terms is different from the concept of sequences of terms occurring in a well-formed formula (wff), the later is defined below.

2.1.3 Definition (***Sequence of wff-terms***)

The *Sequence of wff-terms* in a formula F can be obtained as follows:

(a) If $F = P(t_1, t_2, \ldots\ldots, t_n)$ where P is n-place predicate symbol, then it is $(t_1, t_2, \ldots\ldots, t_n)$.

(b) If $F = \sim F_1$, then it is the sequence of terms in F_1.

(c) If $F = (F_1\ b\ F_2)$ for some binary connective b, then it is the sequence of terms in F_2 appended to the sequence of terms in F_1.

(d) If $F = (Q\ x)\ F_1$ where Q is a quantifier, and x is some variable in F_1, then it is the sequence of terms in F_1.

As an example, (x, y, z, x, z) is the sequence of terms in each of the formulae $((P(x) \vee EAT(y, z)) \wedge (R(x) \rightarrow S(z)))$ and $(\forall x)(\exists y)(\exists z)(((P(x) \vee EAT(y, z)) \wedge (R(x) \rightarrow S(z)))$

The concept of *sequence of predicates* is useful in determining a maximal subwff of the first formula of the resolvable pair of formulae, on which resolution can be performed.

2.1.4 Definition (***Sequence of Predicate-Symbols***)

The *Sequence of Predicate-Symbols* in a formula F can be obtained as follow:

(a) If $F = P(t_1, t_2, \ldots\ldots, t_n)$ where P is n-place predicate symbol, then it is P.

If $F = \sim F_1$, then it is the sequence of predicate-symbols in F_1.

(c) If $F = (F_1\ b\ F_2)$ for some binary connective b, then it is the sequence of predicate-symbols in F_2 appended to the sequence of predicate-symbols in F_1.

(d) If $F = (Q\ x)\ F_1$ where Q is a quantifier, and x is some variable in F_1, then it is the sequence of predicate-symbols in F.

As an example,

(P, EAT, R, S) is the sequence of *predicate-symbols* in each of the formulae $((P(x) \vee EAT(y, z)) \wedge (R(x) \rightarrow S(z)))$ and $(\forall x)(\exists y)(\exists z)(((P(x) \vee EAT(y, z)) \wedge (R(x) \rightarrow S(z)))$

2.1.5 Definition *(U-equivalence of two Wffs)*

Two wffs W_1 and W_2 are said to be *U-equivalent* if

(1) propositional structure of W_1 is truth-functionally equivalent in propositional calculus to the propositional structure of W_2, and

(2) set of predicate-symbols occuring in W_1 equals the set of predicate-symbols occuring in W_2, and

(3) for each predicate-symbol say P occuring in W_1 (or W_2) and for each polarity, there is at most one literal in W_1 with predicate name P.

For example

the two formulae

$wff1$: $(\forall x)\ [P(x) \longrightarrow (\exists y)Q(x,y)]$

and

$wff2$: $(\forall v)(\forall z)(\exists u)[\sim P(u) \vee Q(z,v)]$,

are U-equivalent.

However, the following two formulae

$wff3$: $(\forall x)(\exists y)\ (\forall z)[P(x) \vee (\ (Q(y) \wedge R(z)\ )]$

wff4: $(\exists\ t)(\forall\ u)[\ (\ P(c) \vee Q(u)\ ) \wedge (\ P(d) \vee R(t)\ )\ ]$,

where c and d are constants, are not *U-equivalent* , as condition (3) of U-equivalence is violated.

The concept of structural equivalence helps us recognize when two sub-wffs are unifiable.

2.1.6 Definition (*Strict S-equivalence*)

Two formulae F, G are said to be strictly S-equivalent in all of the following cases.

(a) F and G are atoms, and have the same predicate symbol,

(b) $F = \sim F_1$, $G = \sim G_1$ and F_1, G_1 are strictly S-equivalent,

(c) $F = (F_1\ \ b\ \ F_2)$, $G = (G_1\ \ b\ \ G_2)$ for any binary connective b, and F_1, G_1 and F_2, G_2 are strictly S-equivalent,

(d) $F = [Q\ x]\ F_1$, $G = [Q\ x]\ G_1$ where Q is some quantifier and x is a variable, and F_1, G_1 are strictly S-equivalent. $[Q\ x]$ represents optional quantifier.

2.1.7 Definition (*Reducible S-equivalence*)

If there exists a finite sequence of transformations, from the following set, which when applied to a formula F produces F_1 and/or a formula G produces G_1 (referred to S-equivalent form) such that F_1, G, or F, G_1, or F_1, G_1 are strictly S-equivalent, then F, G are *reducibly S-equivalent*.

The set of transformation rules represented as equivalences are:

1. $(\sim(\sim A)) \leftrightarrow A$
2. $(A \rightarrow B) \leftrightarrow ((\sim A) \vee B)$

3. $(A \leftrightarrow B) \leftrightarrow ((A \rightarrow B) \wedge (B \rightarrow A))$

4. $(\sim(A \vee B)) \leftrightarrow ((\sim A) \wedge (\sim B))$

5. $(\sim(A \wedge B)) \leftrightarrow ((\sim A) \vee (\sim B))$

6. $(A \wedge B) \leftrightarrow (B \wedge A)$

7. $(A \vee B) \leftrightarrow (B \vee A)$

8. $(A \wedge (B \wedge C)) \leftrightarrow ((A \wedge B) \wedge C)$

9. $(A \vee (B \vee C)) \leftrightarrow ((A \vee B) \vee C)$

10. $(A \vee (B \wedge C)) \leftrightarrow ((A \vee B) \wedge (A \vee C))$

11. $(A \wedge (B \vee C)) \leftrightarrow ((A \wedge B) \vee (A \wedge C))$.

2.1.8 Definition *(S-equivalence)*

Two formulae are said to be *S-equivalent* if they are either strictly S-equivalent or reducibly S-equivalent.

2.1.9 Definition *(SU-equivalence of two Wffs)*

Two formulae W_1 and W_2 are *SU-equivalent* if

(1) W_1 and W_2 are U-equivalent, and

(2) there is a formula W_3 which is S-equivalent to W_2 (or W_1) such that W_1 (or W_2) and W_3 have identical propositional structures.

Remarks: In the definition of SU-equivalence, the condition (for U-equivalence) of equivalence of propositional structures of W_1 and W_2, becomes redundant in view of the condition (2) above.

To illustrate the concepts of U-equivalence, S-equivalence and SU-equivalence, we consider some examples below.

The formulae

wff1: $(\forall\ x)(\exists\ y)\ [P(x) \rightarrow Q(x,y)]$

and

wff2: $(\forall\ v)(\forall\ z)[\sim(\ \sim Q(z,v)\ \wedge\ P(v)\ )]$,

are SU-equivalent.

The formulae

wff3: $(\forall\ x)\ [P(x) \rightarrow (\exists\ y)Q(x,y)]$

and

wff4: $(\forall\ v)(\forall\ z)(\exists\ u)[\sim P(u) \vee Q(z,v)]$,

are U-equivalent, but not S-equivalent.

However, The formulae

wff5: $(\forall\ x)(\exists\ u)(\exists\ v)\ [\ P(x)\ \vee\ (Q(u)\ \wedge\ R(v))\ ]$

and

wff6: $(\exists\ s)\ (\forall\ y)(\forall\ z)\ )[\ (\ P(s) \vee Q(z)\ )\ \wedge\ (\ P(y)\ \vee R(z)\ )\ ]$,

are S-equivalent but not U-equivalent.

2.1.10 Definition *(Complementary Subwffs)*

Two subwffs S_1 and S_2 occuring in formulae W_1 and W_2 are said to be *complementary* if polarities of S_1 and S_2 in their parent formulae, are opposite.

WFF-resolution can be done on two subwffs only when they are complementary.

In order to define the concept of a term in the context of WFF-resolution and its extensions, we need the concept of dependency information of an existential variable, which we give below.

2.1.11 Definition (*Dependency Information of an existential Variable*)

For a given formula F, the Dependency information (or dependencies) of an existential variable y (taking polarity into account), occurring as a term of some literal of F or of a function in F, is the set of those variables which are universally quantified (taking polarity into account) in F and in the scope of which y lies in the formula, ***scope being minimum possible after using commutativity of conjunction and disjunction on logically equivalent formulae and using the rules for reducing the scope of quantifiers to the minimum, while again maintaining the logical equivalence of the formulae.***

To illustrate the concept we consider some **examples.**

(1) In each of the two formuale

$$(\forall x)P(x)\ b\ (\exists y)Q(y)$$

and

$$(\forall x)(\ (P(x)\ b\ (\exists y)Q(y)\ ),$$

where b is a binary operation in the set $\{\wedge \vee\}$, the dependency of the only existentially-quantified variable y is nil, because y does not lie in the minimum scope of x. The reason being that the second formula above could have been written equivalently as the first of the two.

(2) Even if, b had been any one of $\rightarrow$ or $\leftrightarrow$, then also the dependency of y would have been nil. In the case of $\leftrightarrow$, we get the fact through using commutativity of $\leftrightarrow$. In the case of b being equal to $\rightarrow$ also, b does not lie in the minimum scope of x as we can think of *A implies B* and *B if A* equivalent, though the order of

occurrence of A and B in the last two expressions is interchanged.

(3) For the formula

$$(\forall x)[(\exists z)(\forall u) P(z, u) \rightarrow (\exists w) Q(x, w)],$$

dependencies of existential variable u is (z) and that of w is (x), because the formula could be written equivalenty (in FOPC) as

$$[(\exists z)(\forall u) P(z, u) \rightarrow (\forall x)(\exists w) Q(x, w)].$$

Remarks: According to this definition of dependencies, the algorithm (given below) for computing dependencies becomes simpler to understand and more efficient. And more significantly, it returns smaller dependencies because now an existential variable is not made to depend on extra variables just because these happen to occur to the left of the existentially-quantified variable because of syntactic compulsions, in the sense that out of two components of a formula, one has to appear before the other.

Also, the use of this method for computing dependencies improves the overall efficiency of the theorem-prover, because now existential variables are functions of smaller number of variables and hence lesser number of variables would be required to be eliminated for resolutions leading to *FALSE* to prove a theorem.

Actually, the method adopted in [Bha 88] is just a legacy of Robinson's method [Rob 60], where in order to remove quantifiers, we shift them first to the left and then we replace the existential-quantified variables by Skolem functions. By adopting this scheme of including only those universal variables in the Skolem function corresponding to an existential variable y, in the minimum scope of which y lies, even Robinson's method can be made more effcient.

Algorithm for computing dependencies of an existential variable y in a formula say F is as follows:

1. Initially take dependencies of the existential variable y as nil.

2. If F is atomic then return dependency of y.

3. If, $F = \sim F_1$, then it is the dependencies of y in F_1, where now quantifications of all the variables have been toggled.

4. If $F = (Q\ x)\ F_1$, where Q is in set $(\forall\ \exists)$, then

(i) if either x is existentially-quantified, taking the polarity into account or x does not occur in F_1, or y does not occur in F_1, then dependencies of y in F is same as dependencies of y in F_1.

(ii) else add x to the dependencies of y in F_1.

5. If $F = F_1\ b\ F_2$, then it is the set union of the dependencies of y in F_1 and in F_2.

For the purpose of matching and unification, an existential variable ν with dependencies $(t_1,t_2,\ldots\ldots,t_n)$, is treated and written as an n-place function-symbol $\nu(t_1,t_2,\ldots\ldots,t_n)$.

The polarity of a quantifier is defined in exactly the same way as *polarity of a subwff* was defined. A quantified variable within the scope of an equivalence relation, to be called *dually quantified*, is both universally as well as existentially

quantified.

Next, we define the notion of a *term* in the context of wff-resolution and its extensions. In order to be specific, we would refer to a term occuring in a well-formed formula as wff-term whereas to a term in the context of the WFF-resolution method as a *resolution-term* (because these terms are defined for the purpose of resolution in wff-resolution method).

2.1.12 Definition (*Term/resolution-term*)

A *resolution-term* (or simply *term*) is defined recursively as follows:

(i) A constant is a term.

(ii) A universal variable is a term.

(iii) An existential variable is a term whenever its dependencies are terms.

(iv) If f is an n-place function symbol, and $t_1,t_2,\ldots\ldots,t_n$ are terms, then $f(t_1,t_2,\ldots\ldots,t_n)$ is a term.

(v) All terms are generated by applying the above rules.

2.1.13 Definition (*Quantified Term of a Term*)

The quantified term of a term t occuring in a closed formula, denoted by $(Q\ t)$, is defined recursively as follows.

(a) If t is a constant, then it is t itself.

(b) If t is a universal variable taking polarity into account, then it is $(\forall\ t)$.

(c) If t is an existential variable taking polarity into account, with dependency information $(t_1,t_2,\ldots\ldots,t_n)$, then it is $(\exists\ t(t_1,t_2,\ldots\ldots,t_n))$.

(d) If $t = f(t_1,t_2,\ldots\ldots,t_m)$, where f is an n-place function-symbol, then it is

$$f(\ (Q_1\ t_1),\ (Q_2\ t_2),\ldots\ldots,(Q_m\ t_m)\ ),$$

where $(Q_i\ t_i)$ is quantified term of t_i, for i = 1,2....,m.

We can obtain the sequence of quantified terms in a subwff as described below:

2.1.14 Definition *(Sequence of quantified terms of a closed formula)*

The *sequence of quantified terms* in a formula F, is obtained by first finding the sequence of wff-terms in F *and* then replacing in the sequence so obtained each wff-term by its quantified term in F.

For example,

the sequence of quantified terms of the wff

$$(\forall\ x)\ [\![\ P(x) \vee (\ (\forall\ y)\ Q(x,y) \rightarrow (\exists\ z)\{R(x,z) \wedge (\forall\ u)\ S(z,u)\})]\!]$$

is

$$\big(\ (\forall\ x),\ (\forall\ x),\ (\exists\ y((\forall\ x))\ ),\ (\forall\ x),\ (\exists\ \ z((\forall\ x)),\ (\exists\ \ z((\forall\ x))),\ (\forall\ u)\ \big),$$

which may be denoted in simplified notation as

$$\big(\ x,\ x,\ y(x),\ x,\ z(x),\ z(x),\ u\ \big).$$

Throughout, we would use the last representation for sequences of quantified terms.

In WFF-resolution, for the purpose of unification and resolution we require only sequences of quantified-terms. Therefore, whenever there is no chance of confusion, in stead of *sequence of quantified terms* we would just say *sequence of terms*.

It is not necessary to define sequence of terms for $(\ F_1 \leftrightarrow F_2\ )$, if there are quantifiers in the scope of F_1 or of F_2, because then each quantifier in F_1 and F_2 is dual and hence for unification with another formula having same propositional structure as $(\ F_1 \leftrightarrow F_2\ )$, we cannot determine quantifications.

Next, we consider unifiability of two formulae.

2.1.15 Definition ***(Unifiability of Two Wffs)***

Two formulae F_1 and F_2 are *unifiable* if

(1) F_1 and F_2 are S-equivalent,

(2) for F_3 which is S-equivalent to F_2 (F_1) such that the propositional structures of F_1 (F_2) and F_3 are identical, the sequences of quantified terms of F_1 (F_2) and F_3 are unifiable.

Since the terms (i.e., quantified-terms) to be unified contain quantifiers, this unification is called *Q-unification* . The basic algorithm of [Rob 65] extends to this methodology. However, we need to consider the case of unification when at least one of the two corresponding terms, is an existential term (because, there is no such concept as existential variable/term in clausal form resolution method, existential variables getting replaced by Skolem constants or Skolem functions). However, the case is disposed of in exactly the same manner as the one in which at least one of the two terms to be unified is a functional term, because as mentioned earlier, for the purpose of matching and unification, an existential variable ν with dependencies $(t_1,t_2,\ldots\ldots,t_n)$, is treated and written as an n-place function $\nu(t_1,t_2,\ldots\ldots,t_n)$. As Q-unification is the only unification we would be dealing with in this thesis, so we may refer to it as just unification.

Next, we state the WFF-resolution inference rule.

Consider two variable-disjoint closed formulae W_1 and W_2 of which S_1 and S_2 are respectively subwffs having opposite polarities in the parent formulae, such that S_1 and S_2 are unifiable. If SET_1 is the set of all subwffs of W_1 that have same propositional structure as S_1 and occur in one conjunct of W_1 and with same polarity in W_1 as S_1, similiarly if SET_2 is the set of all subwffs of W_2 that have same propositional structure as S_2 and occur with same polarity in W_2 as S_2. Further, if SE_1 and SE_2 are any non-empty subsets of respectively

SET_1 and SET_2 and if θ is the most general Q-unification that makes all the subwffs in $SE_1\theta$ and $SE_2\theta$ truth-functionally equivalent, then a *WFF-resolvent* of W_1 and W_2 is obtained by

(i) taking the disjunction of the formulae obtained by replacing each member of $SE_i\theta$ in $W_i\theta$, for i = 1,2, by FALSE, if the member of $S_i\theta$ occurs positively in $W_i\theta$, else by replacing $S_i\theta$ in $W_i\theta$ by TRUE, and then

(ii) simplifying the disjunction obtained at step (i) to a truth-functionally-equivalent formula, free from TRUEs and FALSEs.

One uses this inference rule repeatedly to generate WFF-resolvents until FALSE is generated.

Remarks:

(1) In the case of SU-equivalent formulae say W_1 and W_2, the problem of unification of subwffs W_1 and W_2 reduces to the problem of unification of the corresponding pairs of literals in W_1 and W_2, that have same predicate name and occur with same polarity in the respective parent formula. However, in the case of non-SU-equivalent formulae that are unifiable, setting up such a correspondence between literals may be difficult and computationally costly.

(2) In WFF-resolution we factor a formula by unifying *only those* subwffs (satisfying the restrictions in respect of polarity and propositional structure), all of which belong to one conjunct of the formula, because otherwise the method becomes incomplete as is shown by the following

EXAMPLE: For the following unsatisfiable set *U* of formulae

(1) $P(a) \wedge P(b)$

(2) $\sim P(b)$,

if we factor (1) by unifying all subwffs (having propositional structure P) from more than one conjuncts, i.e, in this case, if we factor (1) by unifying P(a) and P(b) and then resolving the resultant of factoring with (2), then we would be getting the resolvent as $\sim(a = b)$, which is not resolvable, thereby not able to establish unsatisfiability of U.

We consider below two examples in support of our argument in the remarks (1) above,

Example1:

Even in the case of the simplest non-SU-equivalent formulae

$$w_1: \ (\forall x)(\exists y)\ (P(x) \ \vee \ P(y)),$$

and

$$w_2: \ (\forall u)(\exists v)\ (P(u) \ \vee \ P(v)),$$

which are unifiable, we would have to consider two syntactic forms of w_1 namely w_1 and

$$w'_1: \ (\forall x)(\exists y)\ (P(y) \ \vee \ P(x)),$$

in order to verify unifiability of w_1 and w_2.

Example2:

Let P_1, P_2, P_3,.....etc denote different occurrences of predicate symbol P in the non-SU-equivalent unifiable formulae:

$$wff1: (\forall x)(\exists y)\ [P_1(x) \ \wedge \ (\ P_2(a) \ \vee \ P_3(y)\ )\ ],$$

and

$$wff2: (\forall u)\ (\exists\ v)\ [\ (\ P_4(u) \ \wedge P_5\ (v)\ ) \ \vee \ \ (\ P_6(a) \ \wedge \ P_7(u)\ )\ ],$$

where a is a constant. The formulae are unifiable, however, computationally it is quite costly to recognize that the literal $P_1(x)$ from $wff1$ should be unified with literals $P_4(u)$ *and* $P_7(u)$ of $wff2$ and literal $P_3(y)$ should be unified with $P_5(v)$.

Hence, we would restrict to resolutions of formulae on only SU-equivalent subwffs.

Let us look at the application of the wff-resolution considering a simple example from [Qui 61]. The english version of the problem is

Premises:

The guard searched all who entered the premises except those who were accompanied by members of the firm,

Some of Fiorecchio's men entered the premises unaccompanied by anyone else,

The guard searched none of the Fiorecchio's men;

Conclusion:

Some of Fiorecchio's men were members of the firm.

We define the following predicates, in order to represent the above problem in First-Order Logic.

$P(x)$: *x is a person that entered the premises,*

$G(x)$: *x was searched by the guard,*

$H(x, y)$: *x was accompanied by y,*

$M(y)$: *y was a member of the firm,*

$FM(x)$: *x was one of Fiorecchio's men.*

The first-order logic translation of the above problem is

(1) $(\forall x)\,[(P(x) \wedge \sim G(x)) \rightarrow (\exists y)\,(H(x, y) \wedge M(y))]$

(2) $(\exists\ x)\ [FM(x) \wedge P(x) \wedge (\forall\ y)(H(x,y) \rightarrow FM(y))]$

(3) $(\forall\ x)\ [FM(x) \rightarrow \sim G(x))$

Conclusion:

(4) $(\exists\ x)\ (FM(x) \wedge M(x))$

Renaming the variables properly and adding functional dependencies of existential variables, the set with negated conclusion is

U_1: $(\forall x_1)\{\ (P(x_1) \wedge \sim G(x_1)) \rightarrow (\exists y_1(x_1))\ (\ H(x_1, y_1) \wedge M(y_1)\ )\ \}$

U_2: $(\exists x_2())\ \{\ FM(x_2) \wedge P(x_2) \wedge (\forall y_2)\ (\ H(x_2, y_2) \rightarrow FM(y_2)\ )\ \}$

U_3: $(\forall x_3)\ \{\ FM(x_3) \rightarrow \sim G(x_3)\ \}$

U_4: $\sim\{\ (\exists x_4)\ (\ FM(x_4) \wedge M_4)\ )\ \}$

The resolution sequence that leads to *FALSE* is

$U_1 \neq U_2 \dashrightarrow R_1$: Resolving on P^- in U_1 and P^+ in U_2 with

θ: $\{\ (\exists x_2()) / (\forall x_1)\ \}$ gives

$(\exists x_2())\ \{\ \sim G(x_2) \rightarrow (\exists y_1(x_2))\ (\ H(x_2), y_1) \wedge M(y_1)\ )\ \}$,

$R_1 \neq U_3 \dashrightarrow R_2$: Resolving on G^+ in R_1 and G^- in U_3 with

θ: $\{\ (\exists x_2()) / (\forall x_3)\ \}$ gives

$(\exists x_2())\ \{\ [\ (\ \exists y_1(x_2)\ )\ (\ H(x_2, y_1) \wedge M(y_1)\ )\ ] \vee \sim FM(x_2)\ \}$

$R_2 \neq U_2 \dashrightarrow R_3$: Resolving on FM^- in R_2 and FM^+ in U_2 with

θ: $\{\}$ gives

$(\exists x_2())\ [\ (\ \exists y_1(x_2)\ )\ (\ H(x_2, y_1) \wedge M(y_1)\ )\ ]$

$R_3 \neq U_4 \dashrightarrow R_4$: Resolving on M^+ in R_3 and M^- in U_4 with

Θ: [$(\exists x_2())$ / $(\forall x_4)$] gives

$(\exists x_2())$ [$\sim FM(x_2)$]

$R_4 \neq U_2$ ---→ R_5: Resolving on FM^- in R_4 and FM^+ in U_2 with

Θ: [ ] gives

FALSE.

Thus, as *FALSE* is reached as a resolvent, the set of formulae U is unsatisfiable.

In the next section, we consider inference rules that constitute extension of the inference rule WFF-resolution and which include equality.

2.2 WRUE RESOLUTION

In WRUE we can resolve two formulae which have two SU-eqivalent subwffs, one from each, with opposite polarities in parent clauses, even if subwffs fail to unify. This is possible on the basis of the following (informal) argument.

If S_1 and S_2 are two SU-equivalent subwffs of respectively W_1 and W_2, occuring with opposite polarities in respective parent formulae, such that S_1 and S_2 are not unifiable. Let S_3 be a formula which has the same propositional structure as that of S_1, and which is S-equivalent to S_2. Collect all pairs of corresponding terms from S_1 and S_3 where unification fails and form a set DS to be called *origin disagreement set*. If each such pair in DS is assumed to be equal, then S_1 and S_2 after being modified suitably under this assumption, become unifiable, making W_1 and W_2 (modified under the assumption that each pair in D

equals) WFF-resolvable. A unifier of the modified forms of W_1 and W_2 is called *a partial unifier* of modified forms of W_1 and W_2. Let R be a WFF-resolvent of W_1 and W_2. This resolvent R is not absolute, but under the assumption that the condition D holds, where D is the conjunction of the equalities, with one equality corresponding to each of the pair of corresponding terms in DS. In other words, we get the resolvent D → R, which we call *WRUE-resolvent* of W_1 and W_2.

Bhatta [Bha 88] has discussed the equality-based resolution rules *WRUE* and *WNRF* in detail and with illustrations. Essentially to capture the notion of disagreement, a number of definitions like that of disagreement set, origin disagreement set, topmost disagreement set are given, each one in the context of a pair of terms, a pair of complementary literals, a set of literals and a set of SU-equivalent subwffs. We give below some examples of these. However, for details, we refer the reader to [Bha 88] or [DiH 86].

Each of WRUE and WNRF is defined in two different forms namely ***open form*** and ***strong form***. In open form, the user is free to choose both the substitution and the disagreement set. In order to guard against generation of useless resolvents, the strong form allows only restricted choice of substitutions and of disagreement sets.

In [Bha 88] completeness of WRUE in open form is established and completeness of WRUE in strong form is conjuctured.

Next, we consider some examples of disagreement sets before starting with formal discussion of WRUE resolution.

2.2.1 Examples of Disagreement Sets

(a) The disagreement sets of the pair of quantified terms f((∀ x),(∃y(∀x))) and f(b,(∃z(∀x))) are

{ f((∀ x),(∃y(∀x))) : f(b,(∃z(∀x))) }

and

{ (∃y(∀x)) : (∃z(∀x)) }, because (∀ x) unifies with b

(b) The only disagreement set of the pair of terms f((∀ x),(∃y(∀x))) and g(b,(∃z(∀x))) is

{ f((∀ x),(∃y(∀x))) : g(b,(∃z(∀x))) }

(c) The disagreement sets of the pair of terms

f[(∃y(∀x)), h(b, g((∃z(∀x)))], f[b, h((∃z(∀x)), g(d))] are

(i) D1 : { f[(∃y(∀x)), h(b, g((∃z(∀x)))] : f[b, h((∃z(∀x)), g(d))] }, the origin disagrement set,

(ii) D2 : { (∃y(∀x)) : b, h(b, g((∃z(∀x)))) : h((∃z(∀x)), g(d)) },

the top-most disagreement set of the pair

(iii) D3 : { (∃y(∀x)) : b, b : (∃z(∀x)), g((∃z(∀x))) : g(d) },

(iv) D4 : { (∃y(∀x)) : b, b : (∃z(∀x)), (∃z(∀x)) : d }.

(d) For the complementary pair of (quantified) literals P((∀ x), (∃y(∀x))) and ~P(b, (∃z(∀x))), the only disagreement set is { (∃y(∀x)): (∃z(∀x)) }.

(e) The disagreement sets of four literals (after replacing terms by their quantified terms)

P(f((∃y(∀x))), (∀ x)), P((∃z(∀x)), d), P(f(e), m), P(g(n), b)

is D5 : { f((∃y(∀x))) : (∃z(∀x)), f((∃y(∀x))) : f(e), d : m,

f((∃y(∀x))) : g(n), d : b }, where (∀ x) unifies with *d*,

and

D6 : { f((∃y(∀x))) : (∃z(∀x)), (∃y(∀x)) : e, d : m, f((∃y(∀x))) : g(n), d:b }.

By changing the order of consideration of literals, we may get some other disagreement sets, basically yielding the same information as do D5 and D6.

In the following text, by *unifying to the maximum extent possible* we mean that we use the *maximum most general partial unifier* (*mgpu*). For example, when we unify the terms $f(x,g(y, a))$, $f(b, g(c, d))$ we use the substitution $\{b/x, c/y\}$ instead of any of the other partial unifiers $\{b/x\}$ or $\{c/y\}$.

Next we come to the formal statement of WRUE.

Remarks: By a positive (negative) equality, we mean an equality which occurs with positive (negative) polarity in its parent formula.

2.2.2 The WRUE Rule of Inference (open form)

Consider the variable-disjoint closed well-formed formulae W and W' of which S and S' are respectively subwffs having opposite polarities in parent formulae such that S and S' are SU-equivalent.

Let ES = { S_1,.....S_k } be the set of all subwffs (including S) of W that are SU-equivalent to S and such that each S_i has same polarity in W as S does, and occur in one conjunct of W. Let UES be a non-empty subset of ES. Further, let Φ be a partial unifier that (partially) unifies the formulae in UES to the maximum extent possible. Let DS denote a disagreement set of UESΦ and CEQ denote the conjunction of the equalities obtained from DS, one equality corresponding to each pair in DS. Then under CEQ, UESΦ reduces to { SΦ }. Similiarly, if ES', UES', Φ',

DS', CEQ' stand for the corresponding entities w.r.t. S' and W', then under CEQ', $\forall ES'\Phi'$ reduces to { $S'\Phi'$ }.

Further, let Θ be a partial unifier of $S\Phi$ and $S'\Phi'$ and D be a conjunction of the equalities to be satisfied as specified by a disagreement set of { $S\Phi$, $S'\Phi'$ }. If D is true then { $S\Phi\Theta$, $S'\Phi'\Theta$ } reduces to $S\Phi\Theta$.

A *WRUE* resolvent of W and W' is obtained from

$$(CEQ \wedge CEQ' \wedge D) \rightarrow (W\{FALSE/S\Phi\Theta\}) \vee (W'\{TRUE/S\Phi\Theta\},$$

if S occurs positively in W

and

$$(CEQ \wedge CEQ' \wedge D) \rightarrow (W\{TRUE/S\Phi\Theta\}) \vee (W'\{FALSE/S\Phi\Theta\}),$$

if S occurs negatively in W,

by adding and dropping appropriate quantifiers.

We illustrate these ideas for getting a WRUE-resolvent through the following

example

If we want to resolve

wff: [(P ∧ (Q x y)) ∧ (P ∧ (Q p q))] → [(P ∧ (Q r s)) ∧ (P ∧ (Q t u))],

with itself, where P and Q are predicate-symbols and x is a universally quantified variable and the rest of the variables are existentially quantified. Then (P ∧ Q) is the structure of the subwffs on which resolution is possible. The set of subwffs having (P ∧ Q) as *structure* and occuring positively in one conjunct of *wff* is {(P ∧ (Q r s))}. Similiarly, the set of subwffs having (P ∧ Q) as structure and occuring negatively in one conjunct of *wff* is {(P ∧ (Q x y)), (P ∧ (Q p q)}. Thus for resolution, we may unify (partially) a non-empty subset of

sub: {(P ∧ (Q r s)), (P ∧ (Q x y)), (P ∧ (Q p q)},

containg at least two elements, at least one of each polarity in *wff*.

Let us unify

su1: {(P ∧ (Q x y)), (P ∧ (Q r s))}.

Then a resolvent of *wff* with itself is given by

res: (y = s) → (~(P ∧ (Q p q))),

where all the varibles in *res* are existentially quantified.

For solution of problems using WRUE, refer to Appendix 1.

Remarks: We know that factoring is a necessary condition for completeness of a resolution method as is shown by the following example.

The following pair constitutes an unsatisfiable set of formulae:

(1) P(x) ∨ P(y)

(2) ~ P(z) ∨ ~P(u) ; where x, y, z, u are universal variable.

However, we can not derive a refutation unless we allow factoring. This happens because of the fact that P(x) ∨ P(y) and P(x) are equivalent in FOPC, but not computationally in the sense that when resolving with ~P(x) the later gives *FALSE* whereas the former gives P(y), which may not be resolvable later.

However on the other hand, we show through the following example, that compulsory full factoring (that is by merging on ***all*** subwffs having same proposional structure and polarity in the parent formula, occuring in a conjunct) **also** leads to incompleteness.

EXAMPLE: The following set of formulae is clearly unsatisfiable:

(1) ~ (P(a) ∧ P(b))

(2) P(c)

(3) a = c

(4) P(b);

where P is a predicate symbol and *a*, *b*, *c* are constants. In order to derive a refutation we may choose to resolve either P(c) or P(b) with (1). If we use the rule of compulsory factoring on all the subwffs having a given propositional structure and occurring with a given polarity in the formula, then a negative equality (a = b) would be introduced which can not be resolved subsequently, (the resolvent in the case of resolution of (1) with P(c) is $\sim((a=b) \wedge (a=c))$ and in the case of resolution with P(b) it is $\sim(a = b)$). Hence, even compulsory full factoring also leads to incompleteness.

This is why we allow the user to supply any non-empty set of the set of all subwffs having a given propositional structure and occuring in one conjunct and having same polarity in the parent formula.

From the above we draw the conclusion that while resolving a pair of formulae, we should choose those subsets of the sets of all subwffs of the formulae, which unify to that maximum extent such that smallest possible disagreement set is generated. Further, the pairs in disagreement set should be apparently resolvable through other formulae.

2.2.3 The WNRF Rule of Inference (Open Form)

Let W be a closed well-formed formula containing at least one equality literal that occurs negatively in W. Let IE be the set of all equalities of W that occur in one conjunct of W and have negative polarity in W. Let E= { $=_1$, $=_2$,....., $=_n$ } be a non-empty subset of IE. Further if θ is an mgpu of the n equalities (after quantification of terms) in E. By CEQ we denote the conjunction of equalities to be satisfied as specified by a disagreement set of Eθ = { $=_1$, $=_2$,....., $=_n$ }θ. If CEQ is true , then Eθ reduces to a single equality set say { = }.

If = has the form $t_1 = t_2$ and Φ is a substitution and D is a conjunction of equalities to be satisfied as specified by a disagreement set of quantified terms of $t_1\Phi$ and $t_2\Phi$, then

A *WFF-Negative Reflective Function (WNRF) factor* of W is obtained from

$$(CEQ \wedge D) \rightarrow W(\Theta\Phi)\ \{\ TRUE/t_1\Phi = t_2\Phi\ \},$$

by adding and dropping appropriate quantifiers.

To elaborate, we consider an **example**.

Let

wff: $[\ (P \wedge (x = y)) \wedge (P \wedge (p = q))] \rightarrow [(P \wedge (r = s)) \vee (P \wedge (t = u))]$,

where P and Q are predicate-symbols and x is a universally quantified variable and all other variables are existentially quantified.

Then the set of negative equalities in *wff* is $\{(x = y), (p = q)\}$. We consider the set of all the equalities in this set for the purpose of unification. A WNRF-factor of *wff* is given by

wnr: $(y = q) \rightarrow \{P \rightarrow [(P \wedge (r = s)) \vee (P \wedge (t = u))]\}$,

where all the variables in *wnr* are existentially quantified.

2.2.4 WRUE-WNRF Deduction and Refutation

Given a set U of formulae, a WRUE-WNRF deduction of a formula R from U is a finite sequence $R_1, R_2, \ldots\ldots, R_n$ such that each R_i is either a formula in U or a WRUE-WNRF resolvent of formulae preceding R_i in the sequence and where R_n is R.

A deduction of FALSE is called a *WRUE refutation* of U.

2.2.5 WRUE Resolution in Strong Form

WRUE resolution in open form, that is , the method with no restriction on the choice of an *mgpu* and of a disagreement set at any stage of refutation, is complete [2Ha 68]. But, this method may generate any number of useless resolvents. In order to guard against generation of useless resolvents, strong form of resolution puts some restrictions on the choice of an *mgpu and of a disagreement set*. The modified rules of inference (in strong form) are described below. For this we require the following

2.2.5.1 Definition (*Viable Disagreement Set*)

For a theory with given set of axioms U, a disagreement set is said to be *viable*, i.e., can possibly participate in a refutation, only if it satisfies the following conditions:

(1) for each negative equality ($s_i = t_i$) corresponding to a pair $s_i : t_i$ in D, we have that for s_i there is a term g that is the argument of a positive equality literal in U such that s_i either unifies with g or matches g on leading function symbol. In the latter case, $s_i : g$ must have a viable disagreement set below the origin disagreement set. The same is true for t_i and some other term h that is the argument of a positive equality literal in U.

(2) in case there are some negative equalities of the form ($s_i = t_i$) corresponding to pairs $s_i : t_i$ in the disagreement set, that do not satisfy the condition above, then there is a substitution unifying s_i with t_i, for each of these negative equalities, thus converting their logical sum to FALSE.

Furthermore, the substitutions used to satisfy the above condition must be compatible so as to form a single composite substitution. The empty disagreement set is always considered viable.

The definition of viability is recursive, and the recursive iteration is finite because each of the terms s_i and t_i has finite nesting of function symbols (the recursion peels these off and thus terminates)

Viability is a necessary but not a sufficient condition that we can prove $s_i = t_i$ from U and thus erase the corresponding negative equality in a resolvent. A WRUE-WNRF resolvent is viable if its disagreement set is viable.

The weakness of the viability test lies in the fact that when U contains the literal $x = y$ or the literals $x = t$ and $y = t'$, then all negative equalities become viable and the filtering effect is lost. In this case, we always resolve to the top-most disagreement set.

EXAMPLE: We illustrate the concept of viability through an example

For the following set of formulae:

(1) f(x) = b

(2) b =c

(3) c =d

(4) d = g(b)

(5) P(f(a))

(6) ~P(g(d))

(7) ~P(g(a)),

where P is a predicate symbol, f, g are function symbols; x is a universal variable and a, b, c, and d are constants.

The disagreement set of the set of terms{(f(a)), (g(d))} of WRUE-resolvable (open form) formulae (5) and (6), is viable, because, for f(a) there is a term f(x) of a positive equality that unifies with f(a) and for g(d) there is a term g(b) in a positive equality d = g(b) such that for the next lower level disagreement set {b : g} of {g(b) : g(d)}, there are positve equalities containg b and d as terms.

But the pair of terms (g(a), f(a)) of the WRUE-resolvable (open form) formulae (7) and (5) are not viable.

2.2.6 Selection of a Substitution

We show through the following example that constant use of *mgpu* and *bottom-most* disagreement set in WRUE-WNRF can actually prevent a refutation.

EXAMPLE

Consider the following unsatisfiable set U of formulae:

1. P(f(x, b)),

2. ~P(f(a, c))

3. f(b, b) = f(a, c).

where *a*, *b*, *c* are constants and *x* is a universal variable.

Resolving 1. and 2., using *mgpu* (most general partial unifier) and the *bottom most disagreement set*, we get the resolvent

res: ~(*b* = *c*).

But *res* can not be erased.

Therefore, only qualified use of *mgpu* as stated below, is required for WRUE and WNRF.

2.2.6.1 The WRUE Unification Rule

In forming the unifier of complementary subWffs S^+ and S^- in WRUE, we unify on a universal variable x except at an occurrence where x is the argument of a function say f[x], such that there is a viable disagreement set of S^+ and S^-, having a negative equality in the form $f[x] =^- f[t]$.

N.B. : f[x] denotes one occurrence of x anywhere in the argument structure of f, and similiarly for t in f[t]; t and x are at corresponding argument positions so that it is possible to unify x with t in S^+, S^-.

We may apply this rule to a specific occurrence of x without regard to any side effect on other occurrences of x in its clause. For an example, refer [pp. 74, Bha 88].

2.2.6.2 The WNRF Unification Rule

When WNRF is applied to $x =^- t$ we unify with t/x to erase this negative equality. In applying WNRF to $f(a_1, a_2,.....,a_n) =^- f(b_1, b_2,.....,b_n)$ and forming the unifier of these opposed terms, we unify on a universal variable x except at an occurrence where x is the argument of an inner function (say, g[x], which appears within the argument list of f), such that there is a viable disagreement set of $f(a_1, a_2,.....,a_n)$, $f(b_1, b_2,.....,b_n)$ having a negative equality of the form $g[x] =^- g[t]$, where square brackets have the meaning explained before.

This is quite similiar to the WRUE unification rule except that we state that

x is the argument of an inner function and not of the outermost function f.

2.2.6.3 The Strong Form of the Inference Rules and Deduction

The two inference rules WRUE and WNRF in strong form can be obtained by incorporating the following modifications in the corresponding rules in the open form:

(1) In stead of an *mgpu* or a *partial unifier* in the open form of a set of unifiable entities in the open form, we take the unifier obtained by WRUE (WNRF) unification rule described above for strong form of WRUE (WNRF) inference rule.

(2) In stead of any disagreement set of a set of formulae in the open form of the corresponding rule, we take the top-most viable disagreement set of the set of formulae.

2.2.7 The Equality Restriction

For further efficiency, the equality restriction imposed on the *RUE* resolution of complementary equality literals [DiH 86], is extended to WRUE

The WRUE resolution of $W\langle s_1 =^+ s_2\rangle$ and $W'\langle s'_1 =^+ s'_2\rangle$ is permitted only if at least one pair in the set of pairs { $(Q_1\ s_1) : (Q'_1\ s'_1)$, $(Q_2\ s_2) : (Q\ s'_2)$, $(Q_2\ s_2) : (Q'_1\ s'_1)$, $(Q_2\ s_2) : (Q'_2\ s'_2)$ } of corresponding quantified terms unifies or matches on leading-function-symbol. In the latter case, the pair must have a viable disagreement set below the origin disagreement set.

Hence, to resolve equality literals we must satisfy both

(1) the above equality restrictions and

(2) be able to select a topmost viable disagreement set.

The two conditions appear similiar but they are not the same. In finding a topmost

disagreement of complementary literals $s_1 =^+ s_2$ and $s_1 =^- s_2$, the matching by unification or matching on leading function symbol occurs with other literals of the given set of formulae and not among the terms s_i, s_i', $i = 1, 2$ themselves.

Example

To explain the WRUE resolution method, we shall consider a problem of *elementary Geometry*. Alfred Tarski [Tar 51] uses only two predicates to represent the axioms of geometry in FOL. $B(x, y, z)$ represents *"y is between x and z"* and $L(x_1, y_1, x_2, y_2)$ represents *"the distance from x_1 to y_1 is the same as the distance from x_2 to y_2"*. The following are the axioms of Tarski, but without his full axioms of continuity, which is not easily represented in FOL.

Identity axiom for betweenness:

$$U_1: \ (\forall x)\,(\forall y)\ [\ B(x, y, x) \rightarrow x = y\ ]$$

Transitivity axiom for betweenness:

$$U_2: \ (\forall x)\,(\forall y)\,(\forall z)\,(\forall u)\ [\ (B(x, y, u) \wedge B(y, z, u)) \rightarrow B(x, y, z)]$$

Connectivity axiom for betweenness:

$$U_3: \ (\forall x)\,(\forall y)\,(\forall z)\,(\forall u)\ [\ \{\ B(x, y, z\ ) \wedge B(x, y, u) \wedge \sim(x = y)\ \} \rightarrow \{B(x, z, u) \vee B(x, u, z)\ \}\ ]$$

Reflexivity axiom for equidistance:

$$U_4: \ (\forall x)\,(\forall y)\ L(x, y, y, x)$$

Identity axiom for equidistance:

$$U_5: \ (\forall x)\,(\forall y)\,(\forall z)\ [\ L(x, y, z, z) \rightarrow x = y\ ]$$

Transitivity axiom for equidistance:

$$U_6: \ (\forall x)\,(\forall y)\,(\forall z)(\forall u)\,(\forall v)\,(\forall w)$$

$$[\ L(x, y, z, u) \wedge L(x, y, v, w) \rightarrow L(z, u, v, w)]$$

Pasch's axiom:

U_7: $(\forall t)\ (\forall x)\ (\forall y)\ (\forall z)\ (\forall u)\ [\ \{\ B(x, t, u) \wedge B(y, u, z)\ \}$

$$\rightarrow (\forall w)\ \{\ B(x, w, y) \wedge B(z, t, w)\ \}]$$

Euclid's axiom:

U_8: $(\forall t)\ (\forall x)\ (\forall y)\ (\forall z)\ (\forall u)\ [\ \{\ B(x, u, t) \wedge B(y, u, z) \wedge \sim(x = u)\ \}$

$$\rightarrow (\exists v)(\exists w)\ \{\ B(x, z, w) \wedge B(x, y, v) \wedge B(w, t, v)\}]$$

Five-segment axiom:

U_9: $(\forall x_1)\ (\forall x_2)(\forall y_1)\ (\forall y_2)(\forall z_1)\ (\forall z_2)(\forall u_1)\ (\forall u_2)$

$$[\ \{\ L(x_1, y_1, x_2, y_2) \wedge L(y_1, z_1, y_2, z_2)$$

$$\wedge\ L(x_1, u_1, x_2, u_2) \wedge L(y_1, u_1, y_2, u_2)$$

$$\wedge\ B(x_1, y_1, z_1) \wedge B(x_2, y_2, z_2) \wedge \sim(x_1 = y_1) \wedge \sim(y_1 = z_1)\ \}$$

$$\rightarrow\ L((z_1, u_1, z_2, u_2)]$$

Axiom of segment construction:

U_{10}: $(\forall x)\ (\forall y)\ (\forall v)(\forall u)\ (\exists z)\ [\ B(x, y, z) \wedge L(y, z, u, v)\ ]$

Lower dimension axiom:

U_{11}: $(\exists x)\ (\exists y)\ \exists z)\ [\ \sim B(x, y, z) \wedge \sim B(y, z, x) \wedge \sim B(z, x, y)\ ]$

Upper dimension axiom:

U_{12}: $(\forall v)\ (\forall x)\ (\forall y)\ (\forall z)\ (\forall u)$

$$[\ \{\ L(x, u, x, v) \wedge L(y, u, y, v) \wedge L(z, u, z, v) \wedge \sim(u = v)\ \}$$

$$\rightarrow \{B(x, y, z) \vee B(y, z, x) \vee B(z, x, y)\ ]\ ]$$

Weakened continuity axiom:

U_{13}: $(\forall x_1)\ (\forall x_2)(\forall y_1)\ (\forall z_1)\ (\forall z_2)(\forall u_1)$

$$[\ (\ L(u_1, x_1, u_1, x_2) \wedge L(u_1, z_1, u_1, z_2)\ \wedge B(u_1, x_1, z_1)\ \wedge B(x_1, y_1, z_1))$$

$$\rightarrow\ (\exists v_1)\ [\ L(u_1, y_1, u_1, v_1)\ \wedge B(x_2, v_1, z_2)\ )\ ]\ ]$$

The clausal set for the above problem consists of 55 clauses, 20 from the above axioms and 35 from equality axioms for the above problem [WOL 84]

Let us consider a theorem:

For all points x and y, y is between x and y.

The negation of the above theorem in FOL is given by

U_{14}: $\sim(\forall x)\ (\forall y)\ B(x, y, y)$.

The clausal solution with explicit use of equality axioms requires 10 steps where as the solution by WRUE resolution requires only 3 steps as given below.

Only U_5, U_{10}, U_{14} would be required in resolutions given below. These formulae with functional dependencies substituted would be

U_5: $(\forall x_1)\ (\forall y_1)\ (\forall z_1)\ [\ L(x_1, y_1, z_1, z_1)\ \rightarrow\ x_1 = y_1\ ]$

U_{10}: $(\forall x_2)\ (\forall y_2)\ (\forall v)(\forall u)\ (\exists z(x_2, y_2, u, v)\ )\ [\ B(x_2, y_2, z_2) \wedge\ L(y_2, z_2, u, v)\ ]$

U_{14}: $\sim(\forall x_3()\ )\ (\forall y_3()\ )\ B(x_3, y_3, y_3)$.

The resolutions that lead to *FALSE* are as given below:

$U_{14} * U_{10}$ ----→ R_1:

$(\exists x_3()\ )\ (\exists y_3()\ )\ (\exists z_2(x_2, y_2, u, v)\ )\ \sim(z_2\ =\ y_3)$;

with substitution $\{\ (\exists x_3()\ )/(\forall x_2),\ \ (\exists y_3()\ )/\ (\forall y_2)\ \}$

$U_5 * R_1$ ----→ R_2:

$x_3()\ )\ (\exists y_3()\ )(\exists z_2(x_3, y_3, u, v)\ )(\forall z_1)\ \sim L(y_3, z_2, z_1, z_1)$;

with substitution ($(\exists y_3()\,)/x_1$, $(\exists z_2(x_3, y_3, u, v)\,)/y_1$)

$U_{10} \neq R_2 \dashrightarrow R_3$;

FALSE;

with substitution

($(\exists y_3()\,)/(\forall y_2)$, $(\exists x_3()\,)/(\forall x_2)$, $(\forall z_1)/(\forall u_1)$, $(\forall z_1)/(\forall v_1)$,

$(\forall z_1)/(\forall u)$, $(\forall z_1)/(\forall v)$)

2.3 Binary Resolution vs Non-Clausal Resolutions

We have already discussed some disadvantages of clausal form of resolution and also the advantages of non-clausal resolutions including WFF-resolution and its extensions with equalities. However, there are some advantages of clausal resolution over the non-clausal ones. We consider below some of these.

Most operations on non-clausal formulae including the ones with quantifiers in place, are more complex than the corresponding operations on clauses. The result of a non-clausal resolution operation is less predictable than the result of a clausal resolution operation, in the sense that just by looking at a pair of clauses guessing about the expected resolvent of a pair of clauses is easy, however, in the case of formulae, because of the structural complexity of the formulae, guessing the resolvent to be returned is not that easy. This is an important point when a theorem-proving system selects what operation to perform next on the basis of expected result.

Further, from representation point of view, clauses are easily represented as lists of literals; sublists are appended to form the resolvents. Pointers can be used to share lists of literals between a resolvent and its parent formulae. With

simplification being performed during the formation of a non-clausal resolvent, the appearance of a resolvent may differ substantially from its parents, making structure sharing more difficult.

2.4 SPECIAL CASES

We have already mentioned that equality, because of its properties like reflexivity, symmetry, transitivity and substitutivity, occupies a special place in reasoning. Incorporating these properties of equality implicitly is an essential requirement of a good automated reasoning system. We describe in the next section how our system incorporates these properties and how this affects the number of resolvents of a pair of formulae and the number of wnrf-factors of a formula.

Computing appropriate subwffs for the purpose of resolution and then finding the sequences of quantified terms within the scope of biconditionals is not straightforward and needs extra attention. How we handle biconditionals in our theorem-prover is discussed in § 2.4.2

2.4.1 Equality

First we describe briefly how the theorem-prover incorporates implicitly the properties of equality other than symmetry. Next, we discuss about symmetry and its impact on number of resolvents

(1) *REFLEXIVITY*

In order to establish $((\forall x)(x = x) \rightarrow TRUE)$,

we supply the negation

wff1: $\sim((\forall x)(x = x) \rightarrow TRUE)$

to the theorem-prover. The prover immediately simplifies it to *FALSE*.

In the rest of the cases, we assume that x, y, z....etc are constants only. The cases when at least one of these represents an existential variable or a universal variable can be disposed of similiarly.

(2) *TRANSITIVITY*:

In order to establish transitivity, we supply the negation

$$wff: \sim(((x = y) \wedge (y = z)) \rightarrow (x = z)),$$

of the theorem to be proved. The sequence of resolvents generated may be as follows:

wff+wff ----> res1 : $\sim(y = z)$

wff+res1 -----> res2: FALSE

(3) SUBSTITUTION IN PREDICATES:

To establish this property of equality, for each n-place predicate P, we supply the negation

$$wff: \sim(((x_i = x_0) \wedge P(x_1, \ldots, x_i, \ldots, x_n)) \rightarrow (P(x_1, \ldots, x_0 \ldots, x_n))$$

of the theorm to be proved. THe sequence of the resolvents leading to FALSE may be:

wff+wff ----> res1: $\sim (x_i = x_0)$

res1+wff -----> res2: FALSE

Following exactly the same sequence of steps, we can prove the property of substitution in functions for each n-place function symbol.

Next, we come to the symmetry property of equality. This property does not follow automatically. Rather, we have to include it explicitly, and the cost is

increase in number of resolvents. This is achieved by taking all syntactic-variants of the equalities in the set of subwffs on which resolution of two formulae is required to be performed.

For example, if we want to resolve

wff: $[(P \wedge (x = y)) \wedge (P \wedge (p = q))] \rightarrow [(P \wedge (r = s)) \vee (P \wedge (t = u))]$,

with itself, where P is a predicate-symbol. Then $(P \wedge =)$ is the structure of the subwffs on which resolution is possible. The set of subwffs having $(P \wedge =)$ as *structure* and occuring positively in *wff* is $\{(P \wedge (r = s)), (P \wedge (t = u)\}$. Similiarly, the set of subwffs $(P \wedge =)$ as structure and occuring negatively in *wff* is $\{(P \wedge (x = y)), (P \wedge (p = q)\}$. Thus for resolution, we need to unify any non-empty subset of the set

sub: $\{(P \wedge (r = s)), (P \wedge (t = u)) \quad (P \wedge (x = y)), (P \wedge (p = q)\}$,

taking at least one formula of each polarity.

We choose *sub* itself for unifcation of its members.

The set of equalities occuring in the above set is $\{(r = s), (t = u), (x = y), (p = q)\}$. Corresponding to this set there would be $2^{4-1} = 8$ sets of equalities obtained by interchanging the relative positions of terms in equalities such that the sets of terms to be unified are distinct, treating any two pairs of terms (of equalities), of the form $(x\ y)$ and $(y\ x)$ as same. For each variant we would be getting a set of subwff-variants from *sub* by replacing equalities in *sub* by corresponding equalities in the syntactic-variant of the set of equalities. On unification, each syntactic-variant of the set of subwffs may give rise to a distinct unification-list and subsequently a distinct disagreement set. We would get a resolvent corresponding to each syntactic-variant of the set of equalities, leading to 8 resolvents. However, all of these resolvents may not be distinct.

In general, if we want to resolve two formulae *wff1* and *wff2* such that *struc*, the structure of the subwffs on which resolution is possible includes the equality symbol = and such that there are *m1 subwffs of wff1* which occur with polarity as *st-pol* in *wff1* and have structure *struc*; and m2 subwffs of *wff2* which occur with polarity as opposite of *st-pol* in *wff2* and have structure *struc*. Then the number of resolvents may be upto $2^{m1+m2-1}$

The above is sufficient to build in the entire effect of symmetry. It is claimed in [DiH 86] that the virtue of using this rule instead of including the axiom of symmetry in a theory U, is that it leads to a less redundant application of this axiom while preserving completeness, in addition to the fact that it eliminates the need to store the symmetry-variants of formulae. However, we feel that there are disadvantages also of using the rule, in the sense that now we may be required to generate 2^{m+n-1} WRUE resolvents (if no other mechanism is used to have their effect) instead of just one if all the symmetry-variants are separately available for consideration. In our theorem-prover, we have included implicitly the symmetry property of equality.

2.4.2 THE Biconditional

We tackle the problem of biconditionals in the formulae to be resolved or to be factored, by treating each subwff $(A \leftrightarrow B)$ containing the biconditional as having been replaced by $(A \rightarrow B) \wedge (B \rightarrow A)$. Now, if the the polarity of $(A \leftrightarrow B)$ in the parent formula say *wff* is positive, then using distributivity and associativity of $\wedge$ and $\vee$, we get a conjunction of two subwffs each being obtained by replacing the biconditional in *wff*, by one of the two conditionals. This conjunction is equivalently represented by two formulae each representing one of the two conjuncts. However, if the polarity of $(A \leftrightarrow B)$ is negative in *wff*, then we replace $(A \leftrightarrow B)$ by $(A \rightarrow B) \wedge (B \rightarrow A)$ and get just one formula. After getting

one formula/two formulae for *wff*, we resolve it/these with another formula or factor it/these. Thus, if *bcdi* is the number of biconditionals in the formulae wff_i, i = 1, 2, then the number of resolvents may be upto $2^{bcd1+bcd2}$. Also, the number of wnrf-factors of wff_1 may be upto 2^{bcd1}.

Combining with the results of the previous section, we conclude that if a pair of resolvable formulae *wff1* and *wff2* contains respectively *bcd1* and *bcd2* number of biconditionals and if *mi* is the number of subwffs of wff_i, *i*=1, 2, on which resolution is to be performed such that equality is one of the literals in each subwff, then the number of resolvents may be upto $2^{bcd1+bcd+m1+m2-1}$.

Chapter 3

THE IMPLEMENTATION DETAILS

Divide each problem that you examine into as many parts as you need to solve them more easily.

Descartes: *Oeuvres*, vol. VI, p. 18;
Discours de la Methode, Part II.

This rule of Descartes is of little use as long as the art of dividing ...remains unexplained... By dividing his problem into unsuitable parts,the unexperienced problem-solver may increase his difficulty.

Leibnitz: *Philosophishe Schriften*,
edited by Gerhardt, vol. IV, p. 331.

3.1 MAIN ALGORITHM

This section briefly explains the implementation of the *wff-theorem-prover* using pseudo-code. The next section contains a detailed discussion on the procedures constituting the theorem-prover.

The pseudo-code for the theorem-prover:

1: Load the LISP file *wff-theorem-prover*

1. Automatically loads all the LISP files constituting the theorem-prover.
2. Asks the user to go to another window or to create a new window, give the shell command *wff-updater*, in order to keep all the formulae prompted on the window (to be called *information window*) during the session of theorem-proving.
3. Prompts information on the *execution window* (the window on which the theorem-prover is being run) about possible errors and how to handle these errors.
4. Calls the procedure *wff-resolution*.

1.4 The procedure *wff-resolution*

1. Initializes the global variables,
2. Offers the options to choose for tracing either all the major functions, some functions or none . In the case of second option, it requires the user to supply the names of the functions desired to be traced.
3. Calls the procedure *read-wff-list* to read-in the formulae representing all the hypotheses and a negation of the statement to be proved.
4. Offers the following methods to choose one from, for resolution of pair of formulae:

   (i) WFF-resolution (with complete unification),
   (ii) WRUE (open form),
   (iii) WRUE (strong form).

5. Offers iteratively the options to choose one from the following, until the statement is proved or the user wants to quit unsuccessfully:

   (i) resolving a pair of formulae. If the option is chosen then the procedure *select-proper-pair-and-resolve* is called,
   (ii) factoring a formula. If the option is chosen then the procedure *select-proper-wff-n-compute-nrf* is called,
   (iii) to quit-resolution unsuccessfully.

6. Quits successfully proving a particular statement, if at any stage a resolvent or a wnrf returned is FALSE, or unsuccessfully if the user wants to quit without proving a statement.
7. Offers the option of either attempting to prove another statement by calling the procedure *wff-resolution* or to quit to top-level LISP environment.

1.4.3 The procedure read-wff-list

1. Asks the user to choose one of the following forms of representation, to read the initial formulae in:

   (i) internal form,
   (ii) external form,

(iii) natural language form.

2. Prompts an example formula in the form of reprentation chosen at the previous step.

3. Offers the user to choose from one of the above forms of representations in which the formulae would be prompted on the information-window.

4. Iteratively prompts the user to supply valid formulae until nil is supplied.

If ***expression supplied is not valid as a formula***
then
the user is prompted to supply another expression until an expression valid as a formula is supplied
else
(i) it is converted to internal form, if the form chosen for reading formulae is external or natural language form, through respectively the procedures *internalize-ext-form* or *internalize-nl-form*.
(ii) the expression obtained at the previous step is simplified through the procedure *simplify*.
(iii) the expression obtained at previous step is renamed through the procedure *rename-vars*.
(iv) a name of the form |wff-*| is associated with the formula obtained at step (iii), where * is a natural number.
(v) information about the formula, its name and its *sequence-of-polarized-atoms* is prompted on *information-window*.
endif

5. if the value supplied is nil then read-wff-list is terminated.

1.4.5.1 The procedure *select-proper-pair-and-resolve*

1. Prompts the user to supply the names of a pair of formulae, to be resolved. Information about the pairs already tried for resolution is prompted on the *execution-window*.

2. Calls the procedure *ASWFR* to compute *propositional-structure* of the subwffs on which resolution of the two formula is to be carried out. *ASWFR* returns also the polarity with which the subwffs from the first formula, occur in the first formula.

3. Calls the procedure *all-resolvants* to compute all the resolvents of the pair selected at Step 2.

1.4.5.1.3 The procedure *all-resolvants*

1. Tests for the presence of biconditionals in the pair of formulae to be resolved.

2. **If** *none of the two formulae contains a biconditional*
then
the procedure *all-resolvants-aux* or *make-resolvant* is called, depending on whether equality is a part of the subwffs on which resolution is to be performed or not
else
for each biconditional occuring as a main connective of a subwff say $(s_1 \leftrightarrow s_2)$ having positive polarity in one of

6. Assuming that all the members in the set of subwffs become identical after step 5.; for each of the two formulae and corresponding subsets of subwffs chosen at Step 2., each occurrence of a subwff from the subset is replaced in the corresponding formula by *TRUE* if the polarity of the subwff is negative (or nil) in the parent formula, else is replaced by *FALSE*.

7. Each of the two modified formulae obtained at Step 6., is simplified through the procedure *simplify* to drop *TRUEs* and *FALSEs*

8. Quantifiers are added or dropped appropriately from the disjunction of the two formulae obtained at Step 5, by calling respectively the procedures *append-qtfiers* and *drop-qtfiers*.

9. The formula obtained at Step 6 is simplied and is returned as the required resolvent.

10. A name of the form |res-*|, where * is a natural number is associated with the resolvent derived at Step 9.

11. The resolvent returned at Step 9 is prompted on the *execution-window* and also the resolvent alongwith its associated name and its *sequence of polarized atoms* is prompted on the *information window*.

1.4.5.1.3.4.3.3 *The procedure LSM*

1. For each set of literals (having same predicate-name and occuring with same polarity in parent formula), from one of its arguments viz the set of sets of literals, the procedure *SLSM* is called iteratively to modify the unification list and disagreement set.

1.4.5.1.3.4.3.3..1 The procedure *SLSM*

1. It calls the procedure *SLSU* to compute the unification-list of the set of literals supplied to it as an argument.

2. Substitutions are made through procedure *assoc-list-subst* corresponding to this unification-list in the formulae to be resolved , which are also supplied as arguments and also in the subset of set of subwffs returned by *nand-SES* earlier in Step 1. of the procedure *make-resolvant*.

3. Disagreement set is modified through the disagreements of literals in the set of literals through procedure *choose-literal-dis*.

1.4.5.1..3.4..3.3.1.1 The procedure *SLSU*

1. For each pair of literals, formed by taking the first literal in the set of literals supplied to it with the remaining ones one at a time, the sequences of quantified terms are computed through the procedure *subwff-qt-terms*.

2. Using the method of resolution chosen at Step 1.4.4., the unification list is modified through new unifications yielded by the corresponding quantified terms of the pair selectde at Step 1.

1.4.5.2 The procedure *select-proper-wff-n-compute-nrf*

1. **If** *the set of formulae, initially given or obtained later as resolvents or factors* ***does not contain*** *a formula that has at least one equality with negative polarity in the formula*

then

the procedure is exited with appropriate message

else

a formula containing at least one equality with negative polarity in the formula is selected and the procedure *generate-all-nonrf-factors* is called to return all the factors of the formula

endif

1.4.5.2.1 The procedure *generate-all-nonrf-factors*

1. If there is a biconditional in the formula having an equality in its scope then for each occurrence of the biconditional the formula is replaced by one formula or two formulae as in the case of resolving a pair of formulae. But here replacement is done only when there is an equality in the scope of the biconditional. Then the procedure *generate-all-nonrf-factors* is called for each of the newly generated formulae in stead of the selected one.
2. The set of all negative equalities of the formula is returned through the procedure *set-neg-eqls-of*.
3. Set of all syntactic-variants of the set of negative equalities obtained at Step 2, is generated through the procedure *generate-all-eqs*.
4. For each syntactic-variant of equalities in the set obtained at Step 3, the procedure *nonrf-factor* is called with the formula and syntactic-variant as two of its arguments.

1.4.5.2.1.4 The procedure *nonrf-factor*

1. Offers the user to choose a proper subset of the set of (negative) equalities, for any partial factoring of the formula.
2. The set of negative equalities selected at Step 2, or for that matter any set of negative equalities, is a set of literals all having the same predicate-name and negative polarity in the parent formulae. Hence, instead of the procedure *LSM*, the procedure *SLSM* is directly called to generate disagreement set and unification list of the set of negative equalities selected at Step 1.
3. Replaces all those negative equalities in the formula, which belong to the set selected at Step 1., by *TRUE*.
4. Simplifies the formula obtained at Step 3. to remove *TRUEs* and *FALSEs* from it. This gives the desired nnrf-factor.
5. A name of the form |res-*|| where * is a natural number, is associated with the resolvent obtained at Step 4.
6. The nnrf-factor alongwith its associated name is prompted in the *execution-window*. Also, the nnrf-factor alongwith its associted name and its *sequence-of-polarized atoms* is prompted in the *information-window*.

3.2 THE PROCEDURES

The implementation of the theorem prover, *viz.*, WFF-RESOLUTION with equality is done by defining a number of procedures. The procedures which accomplish some subtask or are similar in nature are grouped together into a file. We describe below the various procedures constituting the implementation on a file-by-file basis. The files are considered in a logical order, rather than in a lexicographic order.

3.2.1 The file "aux"

This file contains procedures which are frequently called by procedures contained in other files. Six major modules of procedures as given below are included in this file.

3.2.1.1: The procedure *set-macro-character* allows the user to give abbreviated forms of internal representations of some basic constituents of well formed formulae. For inputs ?Ux, ?Ex, ?Fx and ?Cx it returns respectively ($univ X), ($exist x), ($fun x) and (($fun x)).

3.2.1.2: The second module of procedures is meant for checking an expression as a valid well formed formula. Some of the procedures included in this module are— *is-valid-fr, is-at-fr, is-equality, is-negated-fr, is-qtfied-fr, is-bi-connected-fr, is-wff-seq-of-terms, is-wff-function-term,* etc. If we replace *fr* by *form* and *at* by *atomic*, then it is clear from the name which property of a formula, the corresponding procedure is intended to test.

3.2.1.3: The next module of procedures defines a valid sequence of terms for unification purposes. There is slight difference between a valid term appearing in a formula (to be referred to as a *wff-term*) and a valid term for the purpose of WFF-resolution (to be referred to as a *resolution-term*). The difference between a *wff-term* and a *resolution-term* is that a *wff-term* is what we mean by a term in FOPC, each of which is also a *resolution-term*. But in resolution terms, in addition to the FOPC variables, we may also have existential variables of the form (*x* (*y* t)) denoting the fact that *x* is an existential variable and the quantifier *x* is within the scope of the universally-quantified variables *y* and *t* and further within the scope of no other universally-quantified variable in the well-formed formula under consideration

3.2.1.4 The procedure *concatenate-symbols* accepts a list of numbers, strings and symbols and returns a symbol obtained by first converting the numbers and strings in the list to the corresponding symbols and then concatenating the list of symbols so obtained.

3.2.1.5 Next, we come to one of the most useful module of procedures. This module through its main procedure *proper-read* provides a mechanism for type-checking and for safe-guarding the system against being forced to exit the theorm-prover due to error in the types of the values supplied by the user. The header of the main procedure is

(defun proper-read (var &optional func-env input-stream eof-error-p eof-value recursive-p).

The parameter *var* above stands for the name of the variable for which a value is required to be supplied by the user. The condition *func* in the parameter *func-env* must be satisfied by the value supplied. The condition

3.2 THE PROCEDURES

The implementation of the theorem prover, *viz.*, WFF-RESOLUTION with equality is done by defining a number of procedures. The procedures which accomplish some subtask or are similar in nature are grouped together into a file. We describe below the various procedures constituting the implementation on a file-by-file basis. The files are considered in a logical order, rather than in a lexicographic order.

3.2.1 The file "aux"

This file contains procedures which are frequently called by procedures contained in other files. Six major modules of procedures as given below are included in this file.

3.2.1.1: The procedure *set-macro-character* allows the user to give abbreviated forms of internal representations of some basic constituents of well formed formulae. For inputs ?Ux, ?Ex, ?Fx and ?Cx it returns respectively ($univ X), ($exist x), ($fun x) and (($fun x)).

3.2.1.2: The second module of procedures is meant for checking an expression as a valid well formed formula. Some of the procedures included in this module are— *is-valid-fr*, *is-at-fr*, *is-equality*, *is-negated-fr*, *is-qtfied-fr*, *is-bi-connected-fr*, *is-wff-seq-of-terms*, *is-wff-function-term*, etc. If we replace *fr* by *form* and *at* by *atomic*, then it is clear from the name which property of a formula, the corresponding procedure is intended to test.

3.2.1.3: The next module of procedures defines a valid sequence of terms for unification purposes. There is slight difference between a valid term appearing in

a formula (to be referred to as a *wff-term*) and a valid term for the purpose of WFF-resolution (to be referred to as a *resolution-term*). The difference between a *wff-term* and a *resolution-term* is that a *wff-term* is what we mean by a term in FOPC, each of which is also a *resolution-term*. But in resolution terms, in addition to the FOPC variables, we may also have existential variables of the form (x (y t)) denoting the fact that x is an existential variable and the quantifier x is within the scope of the universally-quantified variables y and t and further within the scope of no other universally-quantified variable in the well-formed formula under consideration

3.2.1.4 The procedure *concatenate-symbols* accepts a list of numbers, strings and symbols and returns a symbol obtained by first converting the numbers and strings in the list to the corresponding symbols and then concatenating the list of symbols so obtained.

3.2.1.5 Next, we come to one of the most useful module of procedures. This module through its main procedure *proper-read* provides a mechanism for type-checking and for safe-guarding the system against being forced to exit the theorm-prover due to error in the types of the values supplied by the user. The header of the main procedure is

(defun proper-read (var &optional func-env input-stream

eof-error-p eof-value recursive-p).

The parameter *var* above stands for the name of the variable for which a value is required to be supplied by the user. The condition *func* in the parameter *func-env* must be satisfied by the value supplied. The condition *func* may be a predicate like *member*, *<*, *symbolp* etc. or a logical combination of predicates. Further, if a predicate like *member* or *<* requires another argument, then it is provided by *env*

component of *func-env*, otherwise *env* is not specified. Rest of the parameters in *proper-read* are those that appear in the system-defined procedure *read*.

The procedure goes on prompting the user to supply values until the value satisfies *func-env*. An example given below may help in elaborating what *proper-read* accoplishes. The function call

(*proper-read* 'x '(*or zerop* (*member* (10 20 30))))

would prompt on the window the following message:

supply a value for x : w.r.t (*or zerop* (*member* (10 20 30))).

If the user supplies any value (not necessarily an integer value) different from 0, 10, 20, or 30, say 15, then the following message is prompted:

Supply another value, the value 15 *is inappropriate.*

The process continues until a value which is in the set {0, 10, 20, 30} is supplied, and which is then accepted.

3.2.1.6 Finally, we have included in the file the definitions of a number of procedures like *our-set-union*, *our-member* etc. which rename some of the system-defined procedures, or which generalize some of the standard procedures. For an example of the later type, the procedure *d-member* (*for deep-member*) with its header

(*defun d-member* (*ele lis*),

determines whether *ele* is a member of *lis* treated as a tree structure, or not. In other words, whether *ele* is either a member of *lis* or a member of a member of *lis* and so on, or not.

3.2.2 The File Convert

In order to facilitate reading and supplying of the formulae by the user, we

have made provision for two other forms of representation of formulae. The two forms are named *EXT* (for *external-form*) and *NLF* (for natural-language form), whereas the form of representation used for the system to manipulate formulae is called *INT* (for internal form). The *INT* representation of formulae includes a large number of parentheses. Many of these parentheses are eliminated from the corresponding representations in *EXT* and *NLF*. *Also, some of the terse forms like*

($exist x), ($univ x), and (($fun f) x y)

are respectively replaced by

()E x), ()U x), and f(x y) in *EXT*

and by

(*for-some* x), (*for-all* x) and f(x y)

in *NLF*.

Further, the symbols *AND, OR, NOT, IMPLIES, IF-AND-ONLY-IF* are used in *NLF* to represent logical operators in stead of the corresponding symbols &, v, ~, →, ↔ used in *INT* and *EXT*.

We give below in the three forms, the representations of a formula.

The formula (in FOPC)

(∃ x) (∀ y)(∃ z) (P (x) → [Q (f(x,y)) → (~ T(z))])

is represented in *INT* by

(($exist x) (($univ y) (($exist z) ((P x) → ((Q (($fun f) x y)) → (~ (T z)))))))

in *EXT* by

()E x) ()U y) ()E z) ((P(x)) → (Q(f(x y) → (~ T(z))))

and in *NL F* by

(*for-some* x) (*for-all* y) (*for-some* z) ((P(x) *implies*

(Q (f(x y))) *implies* (~ T(z))))

The following procedures are included in the file *convert*

—*ext-valid-form*, that checks for the validity of a formula in form *EXT*,

—*nl-valid-form*, that checks for the validity of a formula in form *NLF*,

—*externalize-wff*, returns for given representation of a (valid) formula in *INT*, the representation of the formula in form *EXT*,

—*naturalize-wff*, returns for given representation of a formula in *INT*, the representation of the formula in form *NLF*,

—*internalize-ext-form*, converts representation of a valid formula in form *EXT* to representation of the formula in form *INT*,

—*internalize-nl-form*, converts representation of a valid formula in form *NLF* to representation of the formula in form *INT* .

3.2.3 Equivalence Preserving Formula-Modifiers

In order to facilitate manipulation of the formulae by the system, it is desirable to transform (internally) the formulae to some equivalent simpler form (in *FOPC*) . Also, it is essential for applying resolution (any method) to a pair of formulae that these formulae be variable-disjoint. One simple technique of ensuring variable-disjoint requirement is to rename all the variables in the two formulae and then applying resolution to the modified formulae. Three files viz *lexico*, *simp* and *rename* contain procedures that carry out the above-mentioned transformations on formulae.

3.2.3.1 The File *lexico*

The file contains procedures namely *wff-lexico-comm*, *wff-lexico-asso* and *single-op-gen-asso* which make use of commutativity and associativity of the operations &, ∨, ↔, to transform a given formula into an equivalent formula that has better lexicographic ordering for its predicate symbols and then for its

arguments. It would be less costly to compute resolvable subwffs from formulae with improved lexicographic ordering. The procedures *wff-lexico-comm*, *wff-lexico-asso* make use of respectively commutativity and associativity of the operations &, ∨, ↔ to reorder atomic subwffs in the formulae. The procedure *single-op-gen-assoc* rearranges the atomic formulae in the lexicographic ordering, if any one of the operations &, ∨, ↔ occurs in a subwff at least two times and no other operation occurs in that subwff, by making use of associativity of the operation. This procedure is different from *wff-lexico-asso* in the sense that the later modifies the given formulae w.r.t exactly two consecutive occurrences of an operation from the set (&,∨,↔), whereas this procedure can be applied to modify a formula for any number of consecutive occurrences of an operation.

3.2.3.2 The File *simp*

The main procedure in this file is *simplify*. This procedure eliminates from a formula (except the formulae *TRUE and FALSE*) all occurrences of *TRUE*s and *FALSE*s to return an equivalent formula (in *FOPC*). Further, it makes use of rules of logical equivalence to return shortened and simplified equivalent formulae. For an example, the procedure *simplify* returns *TRUE* , on input of the formula

$$wff: \sim [\ \{\sim(\sim P(x) \wedge TRUE) \wedge (P(x) \vee FALSE)\} \wedge (\sim P(x))\ ].$$

3.2.3.3 The File *rename*

The file *rename* includes a number of procedures with the main procedure named *rename-vars*. These procedures together rename the variables in a given formula in the restricted sense described below.

All occurences of a name in a formula representing the same variable, are

replaced by the same new name. Two occurences of a name representing two distinct variables are replaced by two distinct new names. For example, for the formula (in *INT* form of representation)

((($exist x) (P x)) & ($exist x) (Q x)))),

rename-vars returns

((($exist (x . 1) (P (x . 1)) & ($exist (x . 2) (Q (x . 2)))).

The variable-names in the user-supplied forms of formulae must be symbols only. All the names (with no exception) of all the variables in the user-supplied forms of the formulae are replaced by new names. However, for all other forms of formulae which are either obtained by renaming the variables in a formula or which represent resolvents or factors, only the names of universally-quantified variables are replaced by new variables, keeping the names of existentially-quantified variables intact.

All the new names in the renamed forms of formulae are *conses* of the form (x . *n*) where x is a symbol representing the name of a variable in the user-supplied form of a formula and *n* is a natural number. A symbol x in the user-supplied form of the formula is replaced by a cons (x . *i*), where *i* is some least possible positive integer not used so far in renaming any variable at the time of renaming of x. Further, any variable name in a formula, represented by a cons (x . *k*) is replaced by a cons of the form (x . t), where t is the smallest positive integer not used in renaming any variable so far. We consider below two examples to elaborate these ideas.

Example

For user-supplied formula

wff1 : (($exist x) (($univ y) ((P x) → (~ (Q (($fun f) x y))))))

rename-vars returns

wff2 : (($*exist* (*x* . 5)) (($*univ* (*y* . 5)) ((P (*x* . 5)) → (~ (Q (($*fun* f) (*x* . 5) (*y* . 5)))))),

where 5 is the least positive integer available at the time of renaming variables in *wff1* (1 to 4 having been already used in renaming variables in some other formulae earlier).

However, for formula *wff2*, the procedure *rename-vars* returns

wff3 : (($exist (x . 5)) (($*univ* (y . 6)) ((P (*x* . 5)) → (~ (Q (($*fun* f) (*x* . 5) (*y* . 6)))))),

3.2.4 The File *read-wffs*

This file contains procedures that put on a list named *set-of-wffs*, the formulae that are either initially supplied formulae representing the hypotheses or a negation of the theorem to be proved, or that represent resolvents or wnrfs obtained subsequently. Also, after reading-in each formula, a message of one of the following forms depending on the origin of the formula is printed to a file named *set-of-named-wffs* as well as prompted on the screen:

"*wff*-8 .....*names*..... ((*P*) → ((*Q*) → (*R*)))"

or

"*res*-8 .....*names*..... ((*P*) → ((*Q*) → (*R*)))"

or

"*nrf*-8 .....*names*..... ((*P*) → ((*Q*) → (*R*)))".

The file *set-of-named-wffs* with its latest contents is kept prompted on one of the windows throughout the session of theorem-proving. An executable program named *wff-updater* keeps watch on this file and whenever there is an update of the file because of a message for a newly read formula, the change is immediately reflected on the window by *wff-updater*.

For each of the three types of formulae viz initially supplied, a resolvent and a wnrf, there is a procedure named respectively *read-wff-list*, *put-resolvent-on-set-of-named-wffs* and *put-nonrf-on-set-of-named-wffs* for the purpose explained below. For reading-in initial formulae, the procedure *read-wff-list* provides the user with three options for the forms of representations in which the user wishes to supply the initial formulae. The three forms are *internal form*, which is efficient for the machine to manipulate the formulae in, *external form*, which is compact and easier for the human beings to read and supply formulae in, and finally *natural-language form* which is similiar to the external-form with the difference that it has natural language names for quantifiers and logical connectives. The procedure *read-wff-list* first checks the expression supplied by the user for its validity as a well-formed formula. If the expression supplied does not represent a valid well-formed formula the procedure goes on prompting the user to supply another expression until an expression representing a valid well-formed formula is supplied. Next, if the form used for reading-in the formulae is *external form* or *natural-language form*, it calls a relevant procedure from the file *convert* to convert it to a valid formula in *internal form*. Further, the formula in *internal form* is simplified through the procedure *simplify* and finally the procedure *rename* renames the variables in the formula and puts it on list *set-of-wffs* . Also, if the formula happens to be a positive equality then *read-wff-list* puts this formula on the list named *set-of-positive-equalities*.

The procedures *put-resolvent-on-set-of-named-wffs* and *put-nonrf-on-set-of-named-wffs* put respectively a resolvent and a wnrf on the list *set-of-wffs* and a corresponding message on *set-of-named-wffs*. Further, if resolvent or wnrf happens to be a positive equality then it is put on the list of *set-of-positive-eqs* also.

3.2.5 Auxiliary-Information Yielding Procedures

Next, we consider the file *tot-pol* containing a module of procedures which yield different pieces of inforamation about a formula and its subwffs. The information provided by these procedures is used in other procedures, particularly in the procedures which decide on which subwffs of two given formulae a resolution method is to be applied. We describe below some major procedures in the file, alongwith an example of each for illustration.

3.2.5.1 The Procedure *Polarize*

For a given formula *wff1*, this procedure provides the polarity in *wff1* of each of its subwff . For example, for the formula

wff1 : (($*exist* x) ((P x) → (Q y)))

polarize returns

exp1 : (+ (($*exist* x) (+((-(P x)) → (+(Q y)))))).

The expression *exp1* is interpreted as follows to yield the required information :

Start reading *exp1* from left-hand-side, whenever we get a + or a - sign then it gives the polarity of the subwff obtained by removing all the +s and -s from the subexpression of *exp1* obtained from *exp1* by starting with the left-parenthesis immediately next to the + or - under consideration and continuing upto the matching right plarenthesis.

3.2.5.2 The Procedure *PNAOQ* (*set-of-pol-atoms-wff*)

For a given formula *wff1*, the procedure returns a list of two objects viz

1. the set of all predicate-names occuring in *wff1*, alongwith their polarities in *wff1*. If a predicate-name occurs with both polarities in *wff1* ,then it is in the set

with both polarities,

2. modified formula obtained from the formula after replacing each occurence, if any, of $\mapsto$ with either $\rightarrow$ or $\leftarrow$ suitably.

For example, for the formula

wff2 : (($exist x)(($univ y) (((P x) $\rightarrow$ (P y))) $\mapsto$ ((R y) $\wedge$ ($\sim$ (P y))))),

PNAOQ first prompts the user to supply a value of $\mapsto$. If the value supplied is $\leftarrow$ then *PNAOQ* returns the list

((+P -P -R) (($exist x)($univ y) (((R y) $\wedge$ ($\sim$(P y)))) $\rightarrow$ (((P x) $\rightarrow$ (P y)))))))

3.2.5.3 PSOPA (parenthesized-seq-of-pol-atoms)

For a given formula, *PSOPA* returns a sequence of atoms alongwith their polarities in the given formula and also parentheses are retained to indicate subwff relationship of various components of the formula under consideration. **For illustration**, consider the formula *wff2* of the last subsection alongwith the same replacement $\leftarrow$ for $\mapsto$ in *wff2*. Then *PSOPA* returns

((−P +P) (−R +P))

3.2.5.4 Other Procedures in the Module

The procedures *pred-seq* and *pred-set* return respectively sequence of predicates and set of predicates occuring in a formula. The procedures *arg-seq* and *arg-set* provide respectively the sequence of arguments and set of arguments of atoms and functions occuring in a given formula.

For example,

for the formula *wff2* of §3.2.5.2

pred-seq returns

(P P R P)

pred-set returns

(P R)

arg-seq returns

(x y y y)

and *arg-set* returns

(x y).

3.2.6 The File *deps*

This file represents a module of procedures which either returns dependencies of existential variables of a formula or substitute these dependencies in the formula. The procedure *dependencies* returns for a given formula an association list in which the key is an existential variable and the datum of a key is its dependendency in the given formula. Its auxiliary procedure *dependency-aux*, useful in its own right, returns a list of the following three objects:

1. a list of all the universal variables occuring in the formula,
2. a list of all the existential variables occuring in the formula,
3. an association list of the type mentioned above.

The procedure *dependencies-aux* requires four parameters in its definition namely *subwff1*, *flag1*, *flag2* and *deps-assoc*, where *subwff1* represents a subwff of the given formula say *wff1*, *flag1* indicates the polarity of *subwff1* in *wff1*, *flag2* is a flag which is a non-nil if *subwff1* is in the scope of a biconditional (i.e, $\leftrightarrow$) in the formula *wff1*, else *flag2* is nil and *deps-assoc* is an association-list of the type mentioned above, which is modified by *dependencies-aux* by adding the list of existential-variable-dependency pairs for the existential variables of *wff1*

occuring in *subwff1*. Another procedure in the module is *dependify*, which modifies a given formula by replacing each existential quantifier of the form ($exist x) or ($univ x) (depending upon the polarity of the quantifier in the formula) by respectively ($exist x (y z)) or ($univ x (y z)), where (y z) is the dependency of x in the given formula *wff1*. To explain these ideas, we consider below an **example.**

For the formula

wff5 : (($*exist* x)(($*univ* y) (~(($*exist* z) (($*univ* w) (P x w y z))))),

dependencies-aux returns the list

((y z) (x w) ((w y z))),

where first element (y z) in the list is the set of universal variables in *wff5*, the second element (x w) is the set of existential variables of *wff5* and the last element viz the list ((w y z)) indicates that only existential variable of *wff5* that has non-nil dependency in *wff5*, is *w* and its dependency is (*cdr* (w y z)), *i.e*, (y z).

The procedure *dependencies* returns for the formula *wff5*, the list ((w y z)), the interpretation of which is the same as explained above. Finally, the procedure *dependify* returns for *wff5*, the following formula

(($*exist* x)(($*univ* y) (~(($*exist* z) (($*univ* w (y z)) (P x w y z))))).

3.2.7 The file *Seq*

This file contains a module of procedures with the main procedure *subwff-qt-terms*, that computes for given formula say *wff1* and its subwff say *sub1*, the sequence of quantified terms (for resolution) of any occurence of *sub1* in *wff1*.

For example,

for the formula

wff6 : (($*univ* x) (~(($*univ* y) ((($*univ* z)(P x y z)) → (Q x))))))

and

sub1 : (($*univ* z) (P x y z))),

the procedure call

(*subwff-at-terms* *sub1* *wff6*)

returns

(x (y(x)) z).

The list (x (y(x)) z) denotes that x and z are universal variables and y is an existential variable depending on the universal variable x.

3.2.8 The File *structure*

This file contains three major procedures viz *structure-of*, *structurally-equivalent* and *wang-prove*.

For a given formula, the procedure *structure-of* returns the *propositional-structure* of the formula.

The next procedure *structurally-equivalent* checks for truth-functional equivalence in propositional calculus (PC) of propositional structures of two given formulae. It returns non-nil if the propositional structures are equivalent in PC, otherwise it returns nil.

For example for the two formulae (represented in *INT* form)

wff7: (($*exist* x) ((P x) → ((Q (($*fun* f) x) x)) & (~(R((($*fun* g) x))))))

and

wff8 : (($*univ* y) (((~ (P (($*fun* g) y))) v (~(R y))) & ((P y) → (Q y (($*fun* c))))))),

the procedure *structurally-equivalent* returns *T*. However, if in the formula *wff8*, we replace any one of the the binary logical connective by some other binary connective then the procedure would return *nil*.

The last procedure of the module viz *wang-prove* suggested by Wang [Wan 1960] checks a formula for its being a tautology of *PC*. For a given valid propositional formula *FR*, *wang-prove* returns non-nil if *FR* is a tautology, else it returns *nil*.

3.2.9 The File *apt-for-resolution*

The main procedure *ASWFR* (appropriate-subwff-for-resolution) in the file computes, for two given formulae *wff1* and *wff2*, a maximal propositional structure *PS* for which there exist SU-equivalent subwffs *sub1* and *sub2* respectively of formulae *wff1* and *wff2* such that *sub1* and *sub2* occur with opposite polarities in their respective parent formulae, and also such that propositional structure of *sub1* is *PS*. This procedure is helpful in determining largest possible SU-equivalent (partially) unifiable subwffs of the given (WRUE-) resolvable formulae. For the given two formulae *wff1* and *wff2*, *ASWFR* returns a list of two objects viz the above-mentioned *PS* and the polarity of *sub1* in *wff1*.

For illustration,

we consider below an example. Let

$$wff9 : ((((P) \rightarrow (Q)) \rightarrow (R)) \,\&\, (T))$$

and

$$wff10 : ((S) \,\&\, ((G) \rightarrow ((L) \vee (\sim ((P) \,\&\, (\sim (Q)))))))$$

be two given formulae. Then the call

(*ASWFR wff9 wff10*)

returns

((*P* → *Q*) *nil*).

Note that in the last expression (P → Q) is not a subwff of *wff9* but is a propositional structure of a subwff of *wff9*. The list ((P → Q) *nil*) indicates that there is a subwff of *wff9* with propositional structure (P → Q) that occurs negatively in *wff9*, on which (WRUE-) resolution of *wff9* and *wff10* is possible.

3.2.10 The Various Unification Algorithms

There are three main unification algorithms viz *seq-unify*, *mgp-seq-unify* and *norue* for unifying two sequences of quantified terms. For two formulae *wff1* and *wff2* resolvable on subwffs say *sub1* and *sub2* respectively, the sequences of quantified terms of *sub1* and *sub2* are obtained through the procedure *subwff-qt-terms* contained in the file *Seq*.

3.2.10.1 The procedure *seq-unify*

The procedure *seq-unify* contained in the file *seq-uni* attempts to unify (completely, not partially) the two sequences of quantified terms supplied for unification, by considering possible unification of the terms at corresponding positions in the two sequences. Two sequences of terms for unification are *unifiable* if each pair of terms at corresponding positions in the two sequences, are unifiable. In other words, if unification of any two terms at corresponding positions in the two sequences fails, for the reasons to be explained later, then unification for the two given sequences of terms also fails. In this case, the procedure returns *'$fail*. Also, if the lengths of the two given sequences are not equal then also unification fails and *'$fail* is returned. For two (completely) unifiable sequences of terms, a unification list of pairs is returned, where each pair is a list, the first element of which is a universal variable and the second

element may be either a universal variable, an existential variable or a functional term. Two identical terms at corresponding places in the sequences are taken to be unifiable, but no pair is put on the unification list. Unification fails for two terms at corresponding positions in the two sequences, if out of the two terms, either one is existential term and the other is a functional term or both are functional terms with distinct function names or both are existential terms with distinct names. Two terms, both being existential with same names, are unifiable if their dependencies are unifiable. Similiarly, two functional terms with same function names are unifiable if the lists of their arguments are unifiable. For unification of two terms, it is essential that at least one of the two terms be a universal variable. For a universal variable with more than one occurences in the sequences of terms; after the first occurence of such a variable, unification is attempted between the terms, that correspond to the various occurences of the universal variable, in the other sequence.

This procedure is used for unification of the quantified terms in the WFF-resolution method of inference.

3.2.10.2 The procedure *mgp-seq-unify*

This procedure for unification of pair of sequences of quantified terms is similiar to the procedure *seq-unify* with the difference that *mgp-seq-unify* returns *'$fail* only when the lengths of the two sequences of terms, are not equal. All other cases of the pair of terms for which *seq-unify* returns $fail, are just ignored by *mgp-seq-unify*. After each ignored case of corresponding terms, (partial) unification is attempted for subsequent pairs of terms in the two sequences of terms. The procedure *mgp-seq-unify* returns similiar sequence of terms as does the procedure *seq-unify*.

This procedure is used for unification of the quantified terms in the *WRUE-resolution (open form)* method of inference.

3.2.10.3 The Procedure *ncrue*

This procedure is similiar to the procedure *mgp-seq-unify* for computing partial unifier of sequences of terms (for unification) of two formulae say *wff1* and *wff2* . The procedure *ncrue* differs from the procedure *mgp-seq-unify* only in a special case when corresponding pair of terms to be unified are two distinct functional terms with identical function names. If the two terms are $f(t_1, t_2,.....,t_n)$ and $f(s_1, s_2,.....,s_n)$ such that $s_i \neq t_i$, for at least one i with $1 \leq i \leq n$, then in *mgp-seq-unify*, we attempt partial unification of the sequences of terms for unification that correspond to the argument sequences $(t_1,t_2,.....,t_n)$ and $(s_1, s_2,.....,s_n)$ of the functional terms. However, in *ncrue* we do not attempt unification of $f(t_1, t_2,.....,t_n)$ and $f(s_1, s_2,.....,s_n)$, treating these as pair of not unifiable terms, if the disagreement set of the given formulae *wff1* and *wff2* , which would include the *cons* ($f(t_1, t_2,.....,t_n)$. $f(s_1, s_2,.....,s_n)$), is viable. However, if the above viability requirement is not satisfied then the unification of the two terms $f(t_1, t_2,.....,t_n)$ and $f(s_1, s_2,.....,s_n)$ is carried out as is done by *mgp-seq-unify*.

This procedure is used for unification of the quantified terms in the *WRUE-resolution (strong form)* method of inference.

3.2.11 Computing Disagreements

The file *disagree* contains a number of procedures for computing disagreement set of a given pair of terms, for a given pair of sequences of terms, and for a given set of formulae. Also, there is a procedure in the file namely *generate-n-choose-literal-dis* which, for a given set of formule computes and

returns disagreement set of a particular level desired by the user.

3.2.11.1 The procedure *seq-dis*

For two given sequences of terms for unification, the procedure matches the corresponding terms say s_i and t_i, starting with the first terms of the two sequences. If the corresponding terms match, the procedure is applied to the remaining terms in the two sequences. However, if the two terms s_i and t_i fail to match then a *cons* (s_i . t_i) is formed and is put on (i.e., *consed*) to the disagreement set of the already computed terms.

3.2.11.2 The procedure *literal-dis*

For a given set of formulae, in which each pair of formulae is (*WRUE-*) *unifiable*, the procedure *literal-dis* returns the disagreement set of the formulae in the set. If the set has at most one formula then the disagreement set returned is *nil*. In case there are two formulae in the set, then the sequences of quantified terms for these formulae are computed through the procedure *subwff-qt-terms* and then through a call to the procedure *seq-dis* with the two sequences of terms as arguments, the required disagreement is obtained. Further, if the given set of formulae contains more than two formulae, then disagreement sets of pairs formed by each of the rest of the formulae with the first formula are computed and the union of all these disagreement sets is returned as the required disagreement set.

3.2.11.3 The Procedure *generate-n-choose-literal-dis*

The procedure first computes and prints the *origin-disagreement set* of the given formulae through the procedure *literal-dis*. Then it may compute lower level disagreement sets of the formulae, if so desired by the user. After computation of

disagreement set at each level, the procedure asks the user whether the current disagreement set is the required/ desired one. If the user feels that the current disagreement set is the desired one, then this disagreement set is returned, else disagreement set at next level is computed until lowest level disagreement set is computed. At this stage, the user is given another chance to choose any one of the earlier computed disagreement sets as the one to be returned.

3.2.12 Procedures for Merging

Next, we consider the file named *nand-lit-merge* which contains procedures required for merging SU-equivalent subwffs occuring either in one given formulae all having same polarity in the parent formula or for merging two SU-equivalent subwffs occuring with opposite polarities in two different formulae (WRUE-) resolvable on the two subwffs.

3.2.12.1 Procedure *nand-SES (structurally-equivalent-suwffs)*

The procedure with its header as

(defun nand-SES (wff struc st-pol),

for given a formula *wff*, propositional structure *struc* and polarity *st-pol* returns the set of all subwffs which

(1) have same propositional structure as *struc,*

(2) have polarity *st-pol* in *wff,*

(3) occur in one conjunct of *wff,* assuming occurrences of $\rightarrow$ and $\leftrightarrow$ being replaced by $\wedge$ and $\vee$ maintaining structural equivalence of wff, and $\sim$ being driven deepest possible.

Conditions (2) and (3) are necessary in view of the following observations.

First, in order to show that if condition (2) is violated, then wrue-resolution method is incomplete, we consider the example with the following set of formulae:

F_1: $P(a) \vee \sim P(b)$,

F_2: $P(c)$,

F_3: $\sim P(d)$,

F_4: $b = c$,

F_5: $a = d$,

where a,b,c,d are constants.

It is easily checked that above set is inconsistent. However, if we merge on subwffs which have same predicate-names but may be of opposite polarities then in stead of F_1, we would consider, in any resolution step, its factor

$$F'_1: \quad (a = b) \rightarrow P(a).$$

However, it can be easily checked that we can not deduce *FALSE* from the set $(F'_1, F_2, F_3, F_4, F_5)$ of formulae, because the negative equality (a = b) in F'_1 can not be resolved with any formula in the set or with any resolvent obtained from these formulae.

Next, to show that the condition (3) above is necessary, we consider the following inconsistent set of formulae

F_6: $Q(c) \wedge Q(d)$,

F_7: $\sim Q(f)$,

F_8: $d = f$,

where c, d *and* f are constants.

However, if we factor subwffs from different conjuncts, then in stead of F_6, its factor

F'_6: $(c = d) \rightarrow Q(c)$

would be considered in any resolution step. But, again it is easily checked that we can not deduce *FALSE* from the set $\{ F'_6, F_7, F_8 \}$ of formulae, as the negative equality $(c = d)$ in F'_6 can not be resolved with any of the formula in the set or with any resolvent obtained from these formulae.

For imposing both the conditions (2) and (3) on subwffs, there is some basic reason , that we attempt to explain below. For each formula , consider its skolemized-conjunctive-normal-form (SCNF). Then each conjuct is a clause and hence by (Robinson's) resolution approach, each conjunct can be considered independent of the other conjuncts in SNCF. Thus factoring on subwffs in a formula may be restricted to a conjunct only without affecting the completeness of the WFF-resolution method. However, factoring within a conjunct is necessary as it is in the case of Robinson's method, because Robinson's resolution method is a special case of WFF-resolution method, applied to formulae in which every variable is a universal variable and there is only one conjunct in the formula.

Next, the purpose of computing and then substituting a partial-unifier and then the disagreement set of a set of SU-equivalent subwffs (either for merging within a formula or for resolving two formulae) is to make the subwffs in the set (syntactically) identical through appropriate substitutions provided by the partial unifier and corresponding disagreement set. The partial-unifier-cum-disagreement substitution should be *just sufficient* to achieve the syntactic identity of subwffs in the set, because otherwise we may be introducing some unnecessary inequalities in the resolvents, and these inequalities **may not be resolved later by any resolution steps**. By removing either condition (2) or condition (3), we would be violating this requirement of **just sufficient to make subwffs identical.**

3.2.12.2 Procedure *nand-LISL* (*nand-literals-in-subwff-list*)

Before describing the procedure named above, we briefly state how two or more SU-equivalent subwffs (obtained through the procedure *nand-SES*), can be unified (partially or fully). The unification of the formulae can be carried out on the basis of unification of corresponding literals. For any set of SU-equivalent formulae, we would collect in a set all the literals having same predicate-name and the same polarity in the parent subwff from all the subwffs under consideration. This process is repeated for each possible pair of (*literal-name, polarity*) for the literals occuring in the subwffs of the set under consideration. Thus, we get a set of these sets. Unification (partial or complete) is carried out for each element of this set.

Example:

The procedure *nand-LISL* returns for a given list of subwffs, an association list of the form

(

((+ P) ((P x_1 y_1) (P x_2 y_2) .....)

((- Q) ((Q x_1 y_1) (Q x_2 y_2) .....)

....

....

),

where (+ P) (or (- Q)) indicates that the value of (+ P) in the association list is the the set of all those literals occuring in the given list of subwffs, which have positive (or negative) polarity in the respective subwffs and which have their predicate-name P (or Q).

3.3.12.3 Procedure SLSU (*single-literals-set-unifier*)

The procedure with its header as

(defun SLSU (wff literals-set UM unification-list),

for given a formula named wff, a set viz *literals-set* which is a set of literals having same predicate names and occuring with same polarity in their respective parent subwffs, a unification method *UM* and a unifier *unification-list*, returns modified form of *unification-list* which incorporates in *unification-list*, the possible unification pairs of the form (x t), where x is a universal variable and t is a quantified term of a term occuring in the literals of the *literals-set*.

3.2.12.4 *SLSM (single-literals-set-merge)*

The procedure with its header as

(defun SLSM (wff literals-set UM unification-list disagreement-set),

for given a formula named wff, a set viz *literals-set* which is a set of literals having same predicate names and occuring with same polarity in their respective parent formula, a unification method *UM* and a unifier *unification-list*, and disagreement set *disagreement-set*, returns modified form of *unification-list* as in SLSU and also modified form of *disagreement-set*, where *disagreement-set* is modified adding (*consing*) pairs of quantified terms of the unmatched pairs of the corresponding terms in the modified form of the literals obtained after making, in the literals of the literals-set, substitution provided by (partial) the unifier *unification-list*.

3.2.12.5 Procedure *LSM (literals-sets-merge)*

The procedure LSM, for given a formula *wff* and *set-of-literals-sets* a set of sets of literals, with each set of literals containing literals with same

predicate-name and having same polarity in *wff*, returns a unification-list and a disagreement set as in the case of SLSM. This procedure mainly calls SLSM iteratively until the unification-list and disagreement set are modified for all the sets of set of literals.

3.2.12.6 Procedure *merge-subwffs-aux*

For given a formula *wff*, propositional structure *struc*, polarity *st-pol*, and a unification method *UM*, returns the following three objects:

1. unification-list,

2. disagreement-set,

3. the list of all those subwffs which have propositional structure structurally-equivalent to *struc* and have polarity *st-pol* in *wff*.

The sequence of steps followed by the method is as given below:

(i) this procedure first calls the procedure viz *nand-SES*, to compute the set of all subwffs of *wff*, the propositional structure of which is equivalent to *struc* and which have polarity st-pol in *wff*,

(ii) next, it calls the procedure *nand-LISL* with one of its arguments as the set of subwffs obtained at step (i), to compute the set of the sets of all literals which have same predicate-name and same polarity in the parent subwffs,

(iii) finally, by calling the procedure *LSM* with the set of sets of literals obtained at step (ii) as one of its arguments, it computes the above-mentioned *unification-list* and *disagreement set*.

The terms *unification-list* and *disagreement-set* denote similiar objects as in *LSM*. The set of subwffs is returned so that at the time of resolution, each member of the set may be replaced by *FALSEs* and *TRUEs* appropriately.

3.2.12.7 Procedure *nonrf-merge-eqs-aux*

For a given formula *wff* and a unification method *UM*, the procedure first computes the set of all equalities having negative polarity in *wff*, and then returns, for this set of all equalities occuring in *wff*, the following:

(1) unification-list

(2) disagreement set

(3) the set of all negative equalities occuring in *wff*.

The sequence of steps followed by this procedure is similiar to the sequence of steps followed by the procedure *merge-subwffs-aux*.

Again, the terms *unification-list* and *disagreement-set* denote same objects as they do in the procedures in the previous subsection. The set of negative equalities is computed through another procedure viz *set-neg-eqls-of* defined in this file.

3.2.13 The Substitution Processes

The procedures *LSM* and *merge-subwffs-aux* etc. discussed in the last subsection return for a given set of SU-unifiable subwffs of a formula/formulae, a unification list and a disagreement set. If for each member *mu* of a unification list, each occurence of (*Car mu*) is replaced by (*Cadr mu*) and for each member *md* of a disagreement set, each occurence of (*Car md*) is replaced by (*Cadr md*) in the subwffs of the set, then all the subwffs under consideration become (syntactically) identical. After that the process of factoring a wff or resolving

two resolvable wffs may be carried out further through procedures detailed in later subsections of this section. Thus, we require procedures to implement these operations, of replacing variables in formulae, to be called substitutions.

The procedures for implementing substitutions are spread over three files, each file containing procedures to implement one type of substitution. The substitutions have been divided into three types on the basis of following criteria:

(I) Substitutions due to a unification list returned by LSM or some other procedure, when the set of subwffs supplied to it contains subwffs which are all constituents of one wff (for merging). Procedures of this type of substitution are contained in the file *substi*.

(II) Substitutions due to a unification list returned by LSM or any other procedure, when the set of subwffs supplied to the procedure, contains two U-unifiable subwffs occuring with opposite polarities in two (WRUE-) resolvable formulae. Procedures for this type of substitutions are contained in file *n-substi*.

(III) Substitutions due to a disagreement set, for which procedures are contained in file *dis-sub*.

The three types of substitutions differ from each other only in the matter of replacing the occurences of the quantified variables, i.e., in the matter of the expressions of the form (θ *) where θ ∈{ $exist, $univ } and * is variable.

In the follwing discussion, $\bar{\theta}$ = { $exist, $univ } ~ {θ}, for θ ∈{ $exist, $univ }

Let the list (x t) be a member of a unification list (or the cons (x . t) be a member of disagreement set), where x is a universal variable and t is a term

which is either a universal variable or an existential term but not a functional term. Then a substitution of type I replaces occurences of (θ x) by (θ *) or ($\bar{\theta}$ *) depending upon whether t is a universal variable or existential term where * depends on t. But in case of substitutions of type II and type III, (θ x) is dropped and (θ *) or ($\bar{\theta}$ *) would be placed on a list of quantifiers to be returned. Further, substitutions of type III differ from those of type II in that in the case of substitutions of type III, (θ x) is also placed on the list of quantifiers to be returned whereas (θ x) is not placed on the list of quantifiers for substitutions of type II.

The task of substitution of a term t for a variable x in a formula may be divided into two parts, viz.,

(a) replacing x when x occurs as a term of an atom of the formula,

(b) replacing x in the quantified variable (θ x) by some quantified variable/variables to be determined by t.

In case (a), all three types of substitutions behave identically. In case (b), we have already discussed how they differ from each other.

Next we discuss in detail case (a) where we consider how an occurence x as a term of an atom is affected by the substitution (x t) (or (x . t) for disagreement sets). If t is a universal variable then each occurence of x is replaced by t. If t is an existential term of the form (y (zu)) then x is replaced by the name y of the existential term (y (zu)). If t is a functional term of the form (($fun f) t_1 ... t_n), then x is replaced by (($fun f) s_1 ... s_n) where each s_i is obtained from t_i by replacing each existential term in t_i by the name of the existential term.

To explain the ideas involved we consider some examples:

(i) if (x y) is a pair on unification list, then under the substitution corresponding to this pair the literal $P(x\ s)$ becomes $P(y\ s)$,

(ii) if, in stead of the pair (x y) as in example (i), we have the pair (x $z(y)$) on unification list, then $P(x\ s)$ becomes $P(z\ s)$ after substitution, further

(iii) if, in stead of the pair (x y) as in example (i), we have the pair (x $f(y\ z(u\ v)\quad g(\ w()\ t)\ )\ )$ on unification list, then $P(x\ s)$ becomes $P(\ f(y\ z\ g(w\ t))\ \ s)$ after substitution.

Next, we consider in detail case (b) where we examine how a variable x in a quantified variable (θ x), θ ∈ { $exist, $univ } is affected by a substitution of the form (x t).

First we consider case (b) for substitutions of type I.

If t is a universal variable, then (θ x) is replaced by (θ t). For example, for the substitution (x y), the formula $(\forall\ x)\ P(x\ s)$ becomes $(\forall\ y)\ P(y\ s)$.

If t is an existential term of the form (y (zu)) then (θ x) is replaced by ($\overline{\theta}$ y (zu)). For example, for the substitution (x y(z u)), the formula $\sim(\exists\ x)\ P(x\ s)$ becomes $\sim(\forall\ y(z\ u)\ )\ P(y\ s)$.

If t is a functional term of the form say $f(y\ z(u\ v)\quad g(\ w()\ t)\ )\ )$, then (θ x) is dropped and and each of ($univ y), ($exist z (u v)), ($exist w) and ($univ t) is put on a list of quantifiers, and for the formula $(\forall\ x)P(x\ s)$, the formula $P(\ f(y\ z\ g(w\ t))\ \ s)$ is returned after substitution. The quantified variables in this list of quantifiers are prefixed to whatever expression is obtained after making substitutions for all elements of the unification list.

Next we consider case (b) for substitutions of type II and type III. Here each occurence (θ x) is dropped from the formula. Also only in the case of substitutions of type III, irrespective of the value of θ, the quantifier ($univ x) is put on the list of quantified variables to be returned. Further, if t is a universal variable then ($univ t) is placed on the list of quantified variables. If t is an existential term of the form (y (zu)) then ($exist y (z u)) is placed on the list of quantifiers to be returned. Further, if t is a functional term of the form, say (($fun f) y (z (u v)) (($fun g) (w) t)), then each of ($univ y), ($exist z (u v)), ($exist w) and ($univ t) is placed on the list of quantifiers to be returned. After all the substitutions corresponding to pairs in the given unification list are made, the quantifiers from the list of quantifiers is prefixed to the formula returned earlier.

The procedures named *assoc-list-subst* is the main procedure implementing substitutions of type I. Also procedures named *n-assoc-list-subst* and *dis_list_subst* are the main procedures implementing respectively substitutions of type II and type III.

3.2.14 Procedures for WNRF (Non-Clausal-Negative-Reflectve-Function)

The procedures for computing WNRF of a given formula are contained in the file *nc-nrf*. The procedure *generate-all-ncnrf-factors*, which is the main procedure in the file is discussed next.

3.2.14.1 The Procedure generate-all-ncnrf-factors

The procedure with header

(defun generate-all-ncnrf-factors (name-wff UM)

returns, for given *name-wff*, the name of a formula and a unification method *UM*,

the set of all wnrfs of the corresponding formula. The different wnrfs are generated because, if the formula contains k equalities then there are 2^{k-1} different syntactic variants of these equalities. These variants obtained by interchanging the places of the two terms in an equality are computed by the procedure *generate-all-eqs*

The main procedure first collects into a set all the equalities occuring negatively in the formula with name *name-wff* by calling the procedure *set-neg-eqls-of* and then by calling the procedure *generate-all-eqs* generates all syntactic variants of the equalities in the set. Next, for each syntactic-variant of the equalities it calls iteratively the procedure *nonrf-factor-aux* to compute the corresponding wnrf and puts this wnrf on the set *set-of-nonrf-factors*.

3.2.14.2 The Procedures *nonrf-factor* and *nonrf-factor-aux*

For a given formula *wff* and a unification method UM, the procedure *nonrf-factor* first computes the set of all the equalities occuring negatively in *wff* through a call to the procedure *set-neg-eqls-of* and then gives an option to the user to choose, if desired, any non-empty subset of the above set of equalities, so that wnrf-factors would be computed corresponding to the equalities in the new set. Next, the procedure calls its auxiliary procedure named *nonrf-factor-aux*. The later procedure modifies the the formula *wff* through the following three steps:

1. replaces in the formula *wff* all negative equalities occuring in *wff* by TRUE.
2. the modified formula obtained at step 1. is simplified to a truth-functionally equivalent formula (in FOPC) which is free from TRUEs and FALSEs
3. for all the negative equalities occuring in the originally given formula *wff* a most-general-unifier/mgpu-cum-disagreement-set that makes all the negative equalities identical, is computed through another procedure viz *nonrf-merge-eqs-*

aux, and the substitution provided by most-general-unifier/mgpu-cum-disagreement-set so obtained, is made in the modified form of *wff* obtained at the step 2., to get the finally modified form of *wff*, which is returned by the procedure. Also, the procedure returns the disagreement set obtained at step 3.

In addition to the above procedures, the file *nc-nrf* contains many useful auxiliary procedures. Some of these procedures are described briefly below.

3.2.14.3 Miscellaneous Procedures

For given a formula *wff*, a list of subwffs *subwffs-list* and polarity *st-pol*, the procedure *list-tr-false-subst* replaces in *wff* every occurrence of each member of *subwff-list* by TRUE if *st-pol* is negative; else replaces every occurrence of the members of *subwffs-list* by FALSE.

Another procedure *replace-equalities* replaces in *wff* each equality of the form (= x x) by TRUE.

Further, procedures *add-qtfiers* and *drop-qtfiers* respectively add required quantifiers to and drop redundant quantifiers from the partially computed wnr-f or partially computed resolvent of two given formulae.

3.2.15 Procedures for Resolvents

In this section, we consider the procedures for computing resolvents of two ven formulae. These procedures are contained in the file *all-resols*. In the next osection we consider the main procedure in the file.

3.2.15.1 The Procedure *all-resolvants*

For given the names of two formulae (WRUE-) resolvable on a subwff which has propositional structure say *struc*, and has polarity *st-pol* with which the subwffs which have propositional structure *struc*, occur in the first of the two given formulae, and a unification method *UM*, the procedure *all-resolvants* returns the set of all resolvents of the two formulae. More than one resolvents of the two formulae may be generated because of the same reasons as were given for the generation of more than one wnrfs in §3.2.14.1 or in §2.9.

The procedure first computes, by making calls to procedures *nand-SES* and *nand-LISL*, a set of literals say SL_1 occuring in the subwffs that have propositional structure equivalent to *struc* and have polarity *st-pol* in the first of the two given formulae. Similiarly, it computes another set say SL_2 of literals occuring in the subwffs that have propositional structure *struc* and polarity other than *st-pol* in the second of the two given formulae. If equality literal does not occur in the subwffs , then procedure *make-resolvant* is called to return the list of the only possible resolvent. However, if the subwffs contain equality literals, then the procedure *all-resolvants-aux* is called.

The procedure *all-resolvants-aux*, by making a call to the procedure *generate-all-lits-sets*, computes first the set SSL_1, the elements of which are sets of literals, each obtained from SL_1 by replacing in it some of the equality literals by their symmetry-variants. Similiarly, a set SSL_2 is computed from SL_2. Next, the procedure through calls to procedure *iterate-on-first* and indirect call to procedures *iterate-on-second* and *make-resolvant* , generates all the resolvents of two given formulae and puts them on *set-of-wffs*. Actually *make-resolvant* is directly called by *iterate-on-second* once for each member of

SSL_2. And *iterate-on-second* is called by *iterate-on-first* once for each member of SSL_1.

3.2.15.2 The Procedure *make-resolvant*

The procedure with header

(defun make-resolvant (name-wff1 name-wff2 lits-set-1 lits-set-2 st-pol UM), returns a resolvent of two formulae with names *name-wff1* and *name-wff2*, for given a set of literals viz *lits-set-1* from SSL_1 and a set of literals viz *lits-set-2* from SSL_2, with all literals in *lits-set-1* and *lits-set-2* having same predicate names. The parameters *struc* , *st-pol* and *UM* are used in the same sense as in the last subsection.

The procedure by making calls to the procedure *LSM* computes a (partial) unifier and a disagrement set that make all the literals in *lits-set-1* identical to each other, and a unifier and disagreement set that make all the literals in *lits-set-2* identical to each other. Next, through a call to the procedure *list-tr-false-subst*, it replaces all occurrences of subwffs which have propositional structure equivalent (in PC) to *struc* and occur with polarity *st-pol* in the formula with name *name-wff1*, by TRUEs and FALSEs . suitably. Then from the formula so obtained , all the occurrences of TRUE and FALSE are removed through procedure *simplify* to get a truth-functionally equivalent formula say *nwff1* in *FOPC*. Similiar sequence of steps is followed to get a new formula *nwff2* from the formula with name *name-wff2*, with the difference that in this case the subwffs to be replaced by TRUEs and FALSEs have polarity other than *st-pol* in the formula with name *name-wff2*. Each of *nwff1* and *nwff2* is modified through substiutions provided by the relevant unifier and disagreement set. Also, the first subwff from *lits-set-1* and from *lits-set-2* each is modified through these substitutions to yield new formulae say *new-sub-1* and *new-sub-2* . To make these two formulae identical a (partial) unifier

and a disagreement set are computed, which are used to modify the disjunction *nwff1* ∧ *nwff2* to give a formula which after suitably adding/dropping quantifiers yields the required resolvent.

3.2.16 The File *select-n-resolve*

The file contains two main procedures viz *select-proper-pair-and-resolve* and *select-proper-wff-for-nrf-n-compute*, which respectively select a pair of formulae for resolution and single formula for wnrf. In the next subsection, we discuss the procedure *select-proper-pair-and-resolve*.

3.2.16.1 The procedure *select-proper-pair-and-resolve*

In order to check against attempting at resolving the same pair of formulae again, we keep an association list named *earlier-pairs-assoc* in which each formula say *wff* has its value as the the set of all those formulae which have already been tried for resolution with *wff*. In this context, we have also defined a procedure named *not-earlier-visited* which returns a new pair of formulae, if available, for attempting next resolution on. The procedure *select-proper-pair-and-resolve* selects, if available, a new pair by calling the procedure *not-earlier-visited* and computes the required resolvent. In order to compute the resolvent the main procedure calls the procedure *ASWFR* in order to find the appropriate propositional structure for subwffs of the selected pair of formulae on which resolution may be taken.

3.216.2 The Procedure *Select-proper-wff-for-nrf-n-compute*

Again, in order to check against attempting the computing of the wnrf of the formula more than once, we keep a list named *earlier-tested-for-nrf* of all

those formulae which have been attempted for the purpose of wnrf. The procedure *select-proper-wff-for-nrf-compute* selects through a procedure *viz not-earlier-visited-for-nrf*, a formula hitherto untried for computing wnrf of, and then computes its wnrf.

3.2.17 The File *Integrate*

This is the last but one file to be discussed and the last one to be loaded in the *LISP* environment. It contains, for guiding the user, some messages which are prompted on the window when the file is loaded on the system. Also, there is a procedure named *wff-resolution* defined in this file. We discuss below the procedure.

3.2.17.1 The Procedure *wff-resolution*

The procedure *wff-resolution* is the only procedure which need to be called directly by the user, in order to prove a theorem. .The procedure is called without any arguments.

First of all, the procedure initializes to nil each of the following four lists, to be used in the global sense, namely

1. *set-of-wffs* containing at any stage of execution, all the formulae that represent axioms and a negation of the theorem to be proved, and also the resolvents and wnrfs that have been generated upto that stage.
2. *set-of-positive-equalities* containing all the positive equalities appearing in *set-of-wffs*,
3. *earlier-pairs-assoc* (described in the last subsection) the association list containing at any stage information about pairs of formulae already tried for

resolution,

4. *earlier-tested-for-nnf* the list of all the formulae already tried for wnnf.

Next, the procedure prompts the user, through the procedure *read-wff-list*, to read in the formulae representing all the axioms and a negation of the theorem to be proved. Further, through a loop, it prompts the user iteratively to choose at any stage, one of the following three steps:

1. a resolution step for resolving two formulae,
2. a step for computing wnnf of a formula, and
3. quit-resolution step which allows the user to quit without proving the theorem.

The *only* point to terminate the whole process of resolution, is provided in this procedure. The process of proving the theorem terminates when either

1. the theorem is proved as a result of derivation of either a resolvent or an wnnf which equals FALSE, or
2. the user wants to quit unsuccessfully through the quit-resolution option metioned above.

Finally, irrespective of the two modes of possible termination of the process, the procedure prints all the formulae representing the axioms alongwith the given negation of the theorem to be proved, all the resolvents and the wnnfs

3.2.18 *The file wff-theorem-prover*

This is the *only* file which is required to be loaded *directly by the user* in the lisp environment. The file contains one LISP statement per file for loading each of the other files constituting the theorem-prover. Also, it contains statements that prompt some useful information on one of the windows

CHAPTER 4

CONCLUSIONS AND SCOPE FOR EXTENSION

Human wisdom remains always one and the same
although applied to the most diverse objects and it is
no more changed by their diversity than sunshine
is changed by the variety of objects
which it illuminates

Descarte: Rule I, *Oeuvers*, vol. X, p. 360.

4.1 *Conclusions*

Robinson's resolution method [Rob 60], proved an important step in the area of automated reasoning. However, its language of clauses was a hinderance in its being human-oriented or user-friendly. Later, NC-resolution [Mur 82] improved the situation in this regard to some extent, by requiring only quantifier-free forms keeping the logical connectives in the formulae in place. Finally, WFF-resolution method of Bhatta [Bha 88] uses the formulae without modification.

The theorem-proving system in this thesis implements the WFF-resolution and its variants suggested in [Bha 88], without including sorts.

Though, basically an implementation it includes some new results, which are either improvements of those in [Bha 88] or correct some inaccuracies. In this regard we mention the algorithms about how factoring/merging of a formula is

desirable so that either we reach FALSE faster or we could derive FALSE in some cases where the earlier method could not have derived a FALSE even when the set of formulae was unsatisfiable.

Also, we have improved the definition of *dependencies* of an existential variable, which not only improves the efficiency of computing dependencies but also leads to smaller dependency sets. This is desirable in the sense that in resolutions, we would be required to eliminate fewer variables.

Since the prover is to be used interactively we have to automatically compute the sets of subwffs of the two formulae, on which resolution is to be performed and factoring is to be done. Here an algorithm to recognize some sort of equivalence of subwffs in FOPC is required. In this regard the Wang's algorithm [Wan 60] for proving equivalence of two formulae of propositional calculus proved useful.

Next, we state some general features of the theorem-prover:

1. The system does extensive error checking. In particular user input is thoroughly checked for validity.

2. The system is designed to be user friendly. Whenever there is an error condition, it suggests an appropriate solution to handle the error condition.

Whenever a value is to be supplied of a particular type, it prompts an example of that type in a window.

3. Also to help the user, we have designed it in such a way that all the formulae considered in the resolution process are kept visible throughout the theorem-proving session, on one of the windows. The window, in addition to the formulae also shows their names and also their sequences of polarized atoms. The

last information is useful in selecting an appropriate pair for resolution or for factoring.

4. Further, to help the user in recognizing and supplying the formulae easily, the system offers two more forms - namely *external-form* and *natural-language-form* of representation of the formulae, in addition to the *internal-form*, which is most suited from computational point of view.

5. For enhancing efficiency, we have included some procedures like *lexico* and *simp* to transform given formulae to simpler equivalent forms without disturbing the intuitive aspects of the supplied forms of the formulae.

6. It offers three methods viz WFF-resolution (with complete unification), WRUE (open form) and WRUE (strong form), to choose one from to solve a problem using the theorem-prover.

Thus, we have implemented a procedure which does not require the formulae to be rewritten in terms of the components and which includes the equality relation implicitly.

4.2 Scope for Extensions

It needs to be mentioned that this theorem-prover or for that matter, any theorem-prover is not a problem-solver but just an automated assistant. Our theorem-prover resolves/factors formulae/formula without breaking these/this into its subcomponents and returns a closed formula, which is more comprehensible than a number of clauses representing it.

However, the theorem-prover returns just resolvents and factors. For the

program to be more effective we need some strategies to govern the choice of formula/pair of formulae to be factored/resolved and to reject useless resolvents and factors generated. For this purpose, possible extensions of the theorem-prover may include a strategy similiar to set-of-support strategy for clausal resolution, where set of support may include the special hypotheses of the theory (e,g the four axioms of group theory) alongwith a negation of the theorem to be proved. Also, some strategies like those of demodulation and subsumption for clausal resolution may be included, in order to simplify and canonicalize information that in a sense semantically redundant, although not syntactically redundant can be purged.

In order to make the system more efficient, it may be extended so that many of the subtasks of computing resolvents and factors and/or of computing factors/resolvents for more than one formula/pair of formulae, are carried out in parallel.

Another direction in which the prover can be extended is to design and implement automated provers for other logics, e.g. many-sorted logic, higher-order logics and non-monotonic logics.

APPENDIX 1

HINTS AND EXAMPLES

Reasoning is an art and not a science.
The aphorism applies as well to people as
it does to the computer programs
designed to automate reasoning.

L. Wos

Since arts are more easily learnt by
Examples than Precepts, I have
thought fit to adjoin the Solution of the
following Problems

Newton: *Op.cit.*, pp. 177-178.

In this chapter first we give some hints for efficient use of the theorem-prover and then give examples of some problems solved by using the theorem-prover

APP.1.1 *Hints to User*

It is difficult to state precisely rules in order to solve a problem in the most efficient way using the theorem-prover. However, the following guidelines may

be useful.

1. At the time of representing in FOPC the problem to be solved, we should prefer simple formulae over more complex ones. For a given pair of formulae, one may be simpler than the other if it has lesser number of connectives, and when number of connectives are equal in both, then if it has lesser number of quantifiers. Among the connectives, if intuition does not demand otherwise, we should avoid using the biconditional operator. Use of biconditional operators leads to either more complex resolvents or to proliferation of resolvents or both. For the purpose of counting the number of connectives, for each biconditional we should add three if we add one for any one of the connectives $\wedge$, $\vee$, $\sim$ (because $(A \leftrightarrow B)$ represents $(A \rightarrow B) \wedge (B \rightarrow A)$). Further, the conjunction and disjunction operators should be preferred over conditional operator, again if the intuition does not demand otherwise.

Occurrence of a predicate-name more than once in a formula also complicates the computing of resolvents and wnrf-factors, and hence, if possible, should be avoided.

2. Generally start resolution process by involving the negation of the theorem to be proved.

3. In order to resolve a pair of formulae, again we should prefer simpler formulae over complex ones. This enhances the possibility of resolving to *FALSE* faster. A single-literal formula, if resolvable with another one, would yield a resolvent having lesser number of connectives than the formula it is resolved with.

For each formula read-in, the information-window contains alongwith the formula, its name and sequence of polarized atoms of the formula. We should prefer

a pair of formulae over other pairs, if it has a larger subsequence of the two sequences — one that of the polarized atoms of the first formula and the second which is obtained from the sequence of polarized atoms of the other formula, by replacing each occurrence of plus (+) as a prefix to a predicate-name by minus (-) and vice-versa. In the above we consider subsequences which occur continuously in the the parent sequences..

4. Resolving a formula with one having more terms with universal quantifications in the subwffs to be resolved than another one having more existentially quantified terms in the subwffs to be resolved upon, has better chances of yielding shorter resolvents.

APP.1.2 *Examples with Trace*

APP.1.3 *Examples without Trace*

```
WHENEVER  WE HAVE SOME PROBLEM, WE MAY RESTART RESOLUTION
 BY KEYING-IN THE FOLLOWING LINE:
            (wff-resolution)


You may choose to trace SOME procedures LATER.
At present would you like
to trace ALL of the major procedures?
 if yes then give y else give no
supply a value for WANT-TO-TRACE-ALL : w.r.t  (MEMBER (Y NO))
 INPUT FROM USER --->   y
;;; Loading source file "trace-file"

if the system prompts some message like...
 ERROR: open of..... FAILED ,then give
 (close given-stream)

 WE START WITH READING-IN OF
 THE FORMULAE

Give the formulae representing
 all the hypotheses and
 a negation of the theorem to be proved,
 in one of the following FORMS in which
 you would  like the formulae to be READ
  1.internal-form(INF)
  2.external-form(EXF)
  3.natural-language-form(NLF)

give INF for 1., EXF for 2. and NLF for 3.


 INF is most  uniform of the three representations.Also,it is
 compact, if macro-character ? is used.
 EXF gives formulae in almost mathematical notation
 NLF uses some natural-language elements.
supply a value for FORM-FOR-WFFS :
w.r.t  (OUR-MEMBER (INF EXF NLF))
 INPUT FROM USER --->   inf

An example of representation of a formula in INTERNAL form is:

(
   (($exist x)  (p x y))
              <->
   (
     (
       (~((($univ z)(q   x)))
            ->
         (R z)
```

```
 AND ($exist x) BY ?Ex, USING MACRO-CHARACTER  ?



   VARIABLE      NAMES      MUST    BE    SYMBOLS
  OTHERWISE WE WOULD GET INCORRECT/UNEXPECTED RESULTS


 Would you like to get the formulae prompted
 on the windows in a form DIFFERENT from INTERNAL form
 if so give y else no
supply a value for NEW-OUTPUT-FORM :
w.r.t  (OUR-MEMBER (Y NO))
 INPUT FROM USER --->   no


To end reading originally given wffs,give nil

supply a value for WFF :
w.r.t  IS-VALID-WFF
 INPUT FROM USER --->   (P v)

  The LETTER  v  and any of  the SYMBOLS
        <->  -> &  ~ $fun $univ $exist
      is NOT A VALID VARIABLE / FUNCTION NAME
supply another value , the value (P V) is inappropriate:
 INPUT FROM USER --->   (V s)

  The LETTER  v  and any of  the SYMBOLS
        <->  -> &  ~ $fun $univ $exist
      is NOT A VALID PREDICATE NAME
supply another value , the value (V S) is inappropriate:
 INPUT FROM USER --->   (P x)

The number of atomic formulae of the given formula that are not CLOSED,
hence INVALID equals the number of times this statement appears
supply another value , the value (P X) is inappropriate:
 INPUT FROM USER --->   (($exist x)  (P x))
1 Enter SIMPLIFY (($EXIST X) (P X))
1 Exit SIMPLIFY (($EXIST X) (P X))
1 Enter RENAME-VARS (($EXIST X) (P X))
1 Exit RENAME-VARS (($EXIST (X . 1)) (P (X . 1)))

"|wff-1|   ...names...  (($EXIST (X . 1)) (P (X . 1)))"

To end reading originally given wffs,give nil

supply a value for WFF :
w.r.t (AND IS-VALID-WFF (NOT (OUR-MEMBER (EVAL SET-OF-WFFS))))
 INPUT FROM USER --->   ((Q) -> (R))
1 Enter SIMPLIFY ((Q) -> (R))
1 Exit SIMPLIFY ((Q) -> (R))
1 Enter RENAME-VARS ((Q) -> (R))
1 Exit RENAME-VARS ((Q) -> (R))

"|wff-2|   ...names...  ((Q) -> (R))"

To end reading originally given wffs,give nil
```

```
 INPUT FROM USER --->   (Q)
1 Enter SIMPLIFY (Q)
1 Exit SIMPLIFY (Q)
1 Enter RENAME-VARS (Q)
1 Exit RENAME-VARS (Q)

"|wff-3|    ...names...  (Q)"

To end reading originally given wffs,give nil

supply a value for WFF :
w.r.t (AND IS-VALID-WFF (NOT (OUR-MEMBER (EVAL SET-OF-WFFS))))
 INPUT FROM USER --->   (0
)
supply another value , the value (0) is inappropriate:
 INPUT FROM USER --->   ()
1 Enter SIMPLIFY NIL
1 Exit SIMPLIFY NIL
1 Enter RENAME-VARS NIL
1 Exit RENAME-VARS NIL

Select procedure-number for unification corresponding to the
method of resolution as follows:
   1        for   WFF-resolution  (without partial unification)
   2        for   WRUE (OPEN FORM)
   3        for   WRUE (STRONG FORM)

 supply a value for *METHOD-NUMBER* : w.r.t  (MEMBER (1 2 3))
 INPUT FROM USER --->   2


 would you like next to attempt resolution of two wffs,
 if yes then give y else give no
 supply a value for WANT-TO-RESOLVE : w.r.t  (MEMBER (Y NO))
 INPUT FROM USER --->   y
1 Enter SELECT-PROPER-PAIR-AND-RESOLVE

 for the first wff
w.r.t. its possible selection for next  resolution,
 where in the following (eval set-of-possible-first-names)
is....   (|wff-3| |wff-2| |wff-1|)
supply a value for NAME-OF-WFF1 :
w.r.t  (OUR-MEMBER (EVAL SET-OF-POSSIBLE-FIRST-NAMES))
 INPUT FROM USER --->    |wff-2|

 for the name of second wff
w.r.t. its possible
selection for next  resolution,
where in the following (eval set-of-possible-second-names)
is...  (|wff-3| |wff-2| |wff-1|)
supply a value for NAME-OF-WFF2 :
w.r.t  (OUR-MEMBER (EVAL SET-OF-POSSIBLE-SECOND-NAMES))
 INPUT FROM USER --->   |wff-3|

 The names of two wffs which are to be tried for
 possible resolution are
 |wff-2|
 |wff-3|
```

```
   (Q) Q NIL MGP-SEQ-UNIFY-AUX
|    3 Enter RENAME-VARS ((Q) -> (R))
|    3 Exit RENAME-VARS ((Q) -> (R))
|    3 Enter RENAME-VARS (Q)
|    3 Exit RENAME-VARS (Q)
|    3 Enter NAND-SES ((Q) -> (R)) Q NIL
|    | 4 Enter SMEI (Q) (~ Q) NIL
|    | 4 Exit SMEI (Q)
|    3 Exit NAND-SES ((Q))
|    3 Enter NAND-SES (Q) Q T
|    | 4 Enter SMEI (Q) Q T
|    | 4 Exit SMEI (Q)
|    3 Exit NAND-SES ((Q))
|    3 Enter NAND-LISL ((Q))
|    3 Exit NAND-LISL (((+ Q) (#)))
|    3 Enter NAND-LISL ((Q))
|    3 Exit NAND-LISL (((+ Q) (#)))
          (((+ Q) ((Q))))
|    3 Enter MAKE-RESOLVANT |wff-2| |wff-3| ((Q) -> (R))
       (Q) (((+ Q) (#))) (((+ Q) (#))) Q NIL
       ((Q)) ((Q))
                MGP-SEQ-UNIFY-AUX
|    | 4 Enter LSM ((Q) -> (R)) (((+ Q) (#)))
                MGP-SEQ-UNIFY-AUX
|    |   5 Enter LSM-AUX ((Q) -> (R)) (((+ Q) (#)))
                MGP-SEQ-UNIFY-AUX NIL NIL
|    |   | 6 Enter SLSM ((Q) -> (R)) ((+ Q) ((Q)))
                  MGP-SEQ-UNIFY-AUX NIL NIL
|    |   | 6 Exit SLSM (NIL NIL)
|    |   | 6 Enter LSM-AUX ((Q) -> (R)) NIL
                MGP-SEQ-UNIFY-AUX NIL NIL
|    |   | 6 Exit LSM-AUX (NIL NIL)
|    |   5 Exit LSM-AUX (NIL NIL)
|    | 4 Exit LSM (NIL NIL)
|    | 4 Enter LSM (Q) (((+ Q) (#)))
                MGP-SEQ-UNIFY-AUX
|    |   5 Enter LSM-AUX (Q) (((+ Q) (#)))
                MGP-SEQ-UNIFY-AUX
                  NIL NIL
|    |   | 6 Enter SLSM (Q) ((+ Q) ((Q)))
                MGP-SEQ-UNIFY-AUX
                  NIL NIL
|    |   | 6 Exit SLSM (NIL NIL)
|    |   | 6 Enter LSM-AUX (Q) NIL
                MGP-SEQ-UNIFY-AUX NIL NIL
|    |   | 6 Exit LSM-AUX (NIL NIL)
|    |   5 Exit LSM-AUX (NIL NIL)
|    | 4 Exit LSM (NIL NIL)

TRUE
|    | 4 Enter SIMPLIFY (TRUE -> (R))
|    | 4 Exit SIMPLIFY (R)

FALSE
|    | 4 Enter SIMPLIFY FALSE
|    | 4 Exit SIMPLIFY FALSE
|    | 4 Enter ASSOC-LIST-SUBST NIL (R)
|    | 4 Exit ASSOC-LIST-SUBST (R)
|    | 4 Enter DROP-QTFIERS (R)
|    | 4 Exit DROP-QTFIERS (R)
```

```
|    | 4 Enter DROP-QTFIERS FALSE
|    | 4 Exit DROP-QTFIERS FALSE
|    | 4 Enter ASSOC-LIST-SUBST NIL (Q)
|    | 4 Exit ASSOC-LIST-SUBST (Q)
|    | 4 Enter ASSOC-LIST-SUBST NIL (Q)
|    | 4 Exit ASSOC-LIST-SUBST (Q)
|    | 4 Enter LSM-AUX (((Q) -> (R)) & (Q)) (((+ Q) (#)))
              MGP-SEQ-UNIFY-AUX NIL NIL
|    |   5 Enter SLSM (((Q) -> (R)) & (Q)) ((+ Q) ((Q)))
              MGP-SEQ-UNIFY-AUX NIL NIL
|    |   5 Exit SLSM (NIL NIL)
|    |   5 Enter LSM-AUX (((Q) -> (R)) & (Q)) NIL
              MGP-SEQ-UNIFY-AUX NIL NIL
|    |   5 Exit LSM-AUX (NIL NIL)
|    | 4 Exit LSM-AUX (NIL NIL)
|    | 4 Enter SIMPLIFY ((R) V FALSE)
|    | 4 Exit SIMPLIFY (R)
|    | 4 Enter N-ASSOC-LIST-SUBST NIL (R)
|    | 4 Exit N-ASSOC-LIST-SUBST (R)
|    | 4 Enter DROP-QTFIERS (R)
|    | 4 Exit DROP-QTFIERS (R)
|    | 4 Enter APPEND-QTFIERS NIL (R)
|    | 4 Exit APPEND-QTFIERS (R)
|    | 4 Enter SIMPLIFY (R)
|    | 4 Exit SIMPLIFY (R)
|    | 4 Enter PUT-RESOLVENT-ON-SET-OF-NAMED WFFS (R)
              |wff 2| |wff 3|


THE NEXT RESOLVENT IS GIVEN BY:
"|res-5|   ...names...  (R)"
"             with names of parent-wffs
as ... |wff-2|   and   |wff-3|"
|    | 4 Exit PUT-RESOLVENT-ON-SET-OF-NAMED-WFFS NIL
|    3 Exit MAKE-RESOLVANT (R)
| 2 Exit ALL-RESOLVANTS (R)
1 Exit SELECT-PROPER-PAIR-AND-RESOLVE (R)


 would you like next to attempt resolution of two wffs,
              if yes then give y else give no
 supply a value for WANT-TO-RESOLVE : w.r.t  (MEMBER (Y NO))
 INPUT FROM USER --->   no


 would you like next to attempt ncnrf of a wff,
              if yes then give y else give no
 supply a value for WANT-NCNRF : w.r.t  (MEMBER (Y NO))
 INPUT FROM USER --->   no


 would you like to quit resolution ,if yes
              then give y else give no
 supply a value for WANT-TO-QUIT-RESOLUTION :
              w.r.t  (MEMBER (Y NO))
 INPUT FROM USER --->   y
```

"The set of given wffs alongwith the seq of resolvents is :"

1. !wff-1! ... standing for ... ((*EXIST (X . 1)) (P (X . 1)))
2. !wff-2! ... standing for ... ((Q) -> (R))
3. !wff-3! ... standing for ... (Q)
4. !res-5! ... standing for ... (R)

would you like next to attempt proving ANOTHER THEOREM,
if yes then give y else give no
supply a value for WANT-TO-PROVE-ANOTHER-THEOREM :
w.r.t (MEMBER (Y NO))
INPUT FROM USER ---> no
NIL
: (wff-resolution)

WHENEVER WE HAVE SOME PROBLEM, WE MAY RESTART RESOLUTION
BY KEYING-IN THE FOLLOWING LINE:
(wff-resolution)

You may choose to trace SOME procedures LATER.
At present would you like to trace ALL of the
major procedures
if yes then give y else give no
supply a value for WANT-TO-TRACE-ALL : w.r.t (MEMBER (Y NO))
INPUT FROM USER ---> no

Would you like to trace some procedures?
if so give y else give no.
supply a value for WANT-SOME-TRACING : w.r.t (MEMBER (Y NO))
INPUT FROM USER ---> no

if the system prompts some message like...
ERROR: open of..... FAILED ,then give
(close given-stream)

WE START WITH READING-IN OF
THE FORMULAE

Give the formulae representing
all the hypotheses and
a negation of the theorem to be proved,
in one of the following FORMS in which
you would like the formulae to be READ
1.internal-form(INF)
2.external-form(EXF)
3.natural-language-form(NLF)

give INF for 1., EXF for 2. and NLF for 3.

INF is most uniform of the three representations.Also,it is
compact, if macro-character ? is used.
EXF gives formulae in almost mathematical notation
NLF uses some natural-language elements.

```
 INPUT FROM USER --->   inf

An example of representation of a formula in INTERNAL form is:

(
   (($exist x)  (p x y))
              <->
   (
     (
       (~((($univ z)(q   x)))
            ->
         (R z)
     )
           V
    ( (  S  z x) & (T z) )
   )
)
 IN THE ABOVE WE MAY REPLACE ($univ z) BY ?Uz
 AND ($exist x) BY ?Ex, USING MACRO-CHARACTER  ?



   VARIABLE     NAMES     MUST   BE   SYMBOLS
  OTHERWISE WE WOULD GET INCORRECT/UNEXPECTED RESULTS


 Would you like to get the formulae prompted
 on the windows in a form DIFFERENT from INTERNAL form
 if so give y else no
supply a value for NEW-OUTPUT-FORM :
        w.r.t  (OUR-MEMBER (Y NO))
 INPUT FROM USER --->   no


To end reading originally given wffs,give nil

supply a value for WFF : w.r.t  IS-VALID-WFF
 INPUT FROM USER -       ((P) -> (Q))

"|wff-6|   ...names...  ((P) -> (Q))"

To end reading originally given wffs,give nil

supply a value for WFF : w.r.t
     (AND IS-VALID-WFF (NOT (OUR-MEMBER (EVAL SET-OF-WFFS))))
 INPUT FROM USER --->   (P)

"|wff-7|   ...names...  (P)"

To end reading originally given wffs,give nil

supply a value for WFF :
w.r.t (AND IS-VALID-WFF (NOT (OUR-MEMBER (EVAL SET-OF-WFFS))))
 INPUT FROM USER - ->   (~(Q))

"|wff-8|   ...names...  (~ (Q))"

To end reading originally given wffs,give nil
```

```
 INPUT FROM USER --->   ()

Select procedure-number for unification corresponding to the
      method of resolution as follows:
   1        for   WFF-resolution  (without partial unification)
   2        for   WRUE (OPEN FORM)
   3        for   WRUE (STRONG FORM)

 supply a value for *METHOD-NUMBER* : w.r.t  (MEMBER (1 2 3))
 INPUT FROM USER --->   2


 would you like next to attempt resolution of two wffs,
      if yes then give y else give no
 supply a value for WANT-TO-RESOLVE : w.r.t  (MEMBER (Y NO))
 INPUT FROM USER --->   y

 for the first wff w.r.t. its possible selection for next
      resolution,
 where in the following (eval set-of-possible-first-names)
is....   (|wff-8| |wff-7| |wff-6|)
supply a value for NAME-OF-WFF1 :
w.r.t  (OUR-MEMBER (EVAL SET-OF-POSSIBLE-FIRST-NAMES))
 INPUT FROM USER --->    |wff-6|

 for the name of second wff w.r.t. its possible
selection for next  resolution,
where in the following (eval set-of-possible-second-names)
is...  (|wff-8| |wff-7| |wff-6|)
supply a value for NAME-OF-WFF2 :
w.r.t  (OUR-MEMBER (EVAL SET-OF-POSSIBLE-SECOND-NAMES))
 INPUT FROM USER --->     |wff-7|

 The names of two wffs which are to be tried for
 possible resolution are
 |wff-6|
 |wff-7|
(((+ P) ((P))))
TRUE
FALSE


THE NEXT RESOLVENT IS GIVEN BY:
"|res-10|   ...names...  (0)"
"            with names of parent-wffs
as ... |wff-6|   and   |wff-7|"


 would you like next to attempt resolution of two wffs,
      if yes then give y else give no
 supply a value for WANT-TO-RESOLVE : w.r.t  (MEMBER (Y NO))
 INPUT FROM USER --->   |res-10|
supply another value , the value |res-10| is inappropriate:
 INPUT FROM USER --->   y

 for the first wff w.r.t. its possible selection for next
      resolution,
 where in the following (eval set-of-possible-first-names)
      is....   (|res-10| |wff-8| |wff-7| |wff-6|)
supply a value for NAME-OF-WFF1 :
```

```
 for the name of second wff w.r.t. its possible
selection for next  resolution,
where in the following (eval set-of-possible-second-names)
      is...  (|res-10| |wff-8| |wff-7| |wff-6|)
supply a value for NAME-OF-WFF2 :
w.r.t  (OUR-MEMBER (EVAL SET-OF-POSSIBLE-SECOND-NAMES))
 INPUT FROM USER --->   |wff-8|

 The names of two wffs which are to be tried for
 possible resolution are
 |res-10|
 |wff-8|
(((+ Q) ((Q))))
FALSE
TRUE


THE NEXT RESOLVENT IS GIVEN BY:
"|res-11|   ...names...  FALSE"
"             with names of parent-wffs
as ... |res-10|   and   |wff-8|"


"The given theorem is valid"

"The set of given wffs alongwith the seq of resolvents is :"

  1.  |wff-6|   ... standing for ...  ((P) -> (Q))
  2.  |wff-7|   ... standing for ...  (P)
  3.  |wff-8|   ... standing for ...  (~ (Q))
  4.  |res-10|   ... standing for ...  (Q)
  5.  |res-11|   ... standing for ...  FALSE



 would you like next to attempt proving ANOTHER THEOREM,
      if yes then give y else give no
 supply a value for WANT-TO-PROVE-ANOTHER-THEOREM :
      w.r.t  (MEMBER (Y NO))
 INPUT FROM USER --->   no
NIL
```

SHUBERT'S STEAMROLLER PROBLEM READS AS FOLLOWS:

Wolves, foxes, birds, carerpillars,and snails are animals.
Grains are Plants.
There exist wolves, foxes, birds, caterpillars, snails,
and grains.
Every animal eats all plants or any smaller animal that
eats some plants.
Birds are smaller than foxes which in turn are smaller
than wolves.
Wolves do not eat foxes or grains. Birds eat caterpillars,
but no snails.
Caterpillars and snails eat some plants

THE THEOREM TO BE PROVED:

There is a grain eating animal that is eaten by another
animal.

Let the predicate symbols be defined as below:

WOLF(x)	: x is a wolf,
FOX(x)	: x is a fox,
BIRD(x)	: x is a bird,
CATERPILLAR(x)	: x is a caterpillar,
SNAIL(x)	: x is a snail,
GRAIN(x)	: x is a grain,
PLANT(x)	: x is a plant,
ANIMAL(x)	: x is an animal.
EATS(x,y)	: x eats y
SMALLER(x,y)	:x is smaller than y

The FOPC translation of the above problem, with negated conclusion is

(IN INTERNAL REPRESENTATION):

```
1.      (($univ x)((WOLF   x)              ->  (ANIMAL x) ))
2.      (($univ x)((FOX    x)              ->  (ANIMAL x) ))
3.      (($univ x)((BIRD   x)              ->  (ANIMAL x) ))
4.      (($univ x)((CATERPILLAR  x)        ->  (ANIMAL x) ))
5.      (($univ x)((SNAIL  x)              ->  (ANIMAL x) ))
6.      (($univ x)((GRAIN  x)              ->  (PLANT x) ))
7.      ((((WOLF  ?cLUPO) & (FOX ?cFOXY))
                &  ((BIRD ?cTWEEDY) & (CATERPILLAR ?cMAGGIE)))
                     & ((SNAIL ?cSLIMEY) & (GRAIN ?cSTALKY)))
8.      (($univ w) ((ANIMAL w) ->
            ( ($univ x) (( (PLANT x) -> (EATS w x)) V
                 (($univ y) ( (  (ANIMAL y) & (SMALLER y w))  &
                    ( ($exist z )( ((PLANT z) & ( EATS y z))  ->
                          (EATS w y) ) ) ) ) )) ))
```

```
          (SMALLER x y))))
11.     (?Ux (?Uy (( (BIRD          x) & (FOX y)) ->
          (SMALLER x y))))
12.     (?Ux (?Uy (( (FOX          x) & (WOLF y)) ->
          (SMALLER x y))))
13.     (?Ux (?Uy (( (WOLF          x) & (FOX y)) ->
          (~(EATS  x y)))))
14.     (?Ux (?Uy (( (WOLF          x) & (GRAIN y)) ->
          (~(EATS  x y)))))
15.     (?Ux (?Uy (( (BIRD          x) & (CATERPILLAR y)) ->
          (EATS x y))))
16.     (?Ux (?Uy (( (BIRD          x) & (SNAIL y)) ->
          (~(EATS  x y)))))
17.     (?Ux ( (CATERPILLAR x) ->
          (?Ey ( (PLANT y) & (EATS x y)))))
18.     (?Ux ( (SNAIL x) -> (?Ey ( (PLANT y) &
          (EATS x y)))))
19.     (~( ?Ex (SMALLER x ?CMAGGIE)))
20.     (~ (?Ux (EATS x x)))
21.     (( ?Ux ((ANIMAL x) -> (~(PLANT x))))    &
              ( ?Ux ((~(PLANT x)) -> (ANIMAL x))))
22.     (~ (?Ex (?Ey (  (ANIMAL x) & ((ANIMAL y)
      & ((EATS x Y) & (?Uz ((GRAIN z) -> (EATS y z)))))))))
          ()

"|wff-1|    ...names...
    ((SUNIV (X . 1)) ((WOLF (X . 1)) -> (ANIMAL (X . 1))))"
"|wff-2|    ...names...
    ((SUNIV (X . 2)) ((FOX (X . 2)) -> (ANIMAL (X . 2))))"

"|wff-3|    ...names...
    ((SUNIV (X . 3)) ((BIRD (X . 3)) -> (ANIMAL (X . 3))))"
"|wff-4|    ...names...
((SUNIV (X . 4)) ((CATERPILLAR (X . 4)) -> (ANIMAL (X . 4))))"
"|wff-5|    ...names...
    ((SUNIV (X . 5)) ((SNAIL (X . 5)) -> (ANIMAL (X . 5))))"

"|wff-6|    ...names...
    ((SUNIV (X . 6)) ((GRAIN (X . 6)) -> (PLANT (X . 6))))"

"|wff-7|    ...names...
    ((((WOLF ((SFUN LUPO))) & (FOX ((SFUN FOXY)))
  & ((BIRD ((SFUN WEEDY))) & (CATERPILLAR ((SFUN MAGGIE)))))
      & ((SNAIL ((SFUN SLIMY))) & (GRAIN ((SFUN STALKY)))))"
"|wff-8|    ...names...    ((SUNIV (W . 8)) ((ANIMAL (W . 8)) ->
      ((SUNIV (X . 8)) (((PLANT (X . 8))
       (EATS (W . 8) (X . 8)))
       V ((SUNIV (Y . 8)) (((ANIMAL (Y . 8)) &
      (SMALLER (Y . 8) (W . 8)))
      & ((SEXIST (Z . 8))
      (((PLANT (Z . 8)) & (EATS (Y . 8) (Z . 8)))
                        -> (EATS (W . 8) (Y . 8))))))))))"

"|wff-9|    ...names...
    ((SUNIV (X . 9)) ((SUNIV (Y . 9)) (((CATERPILLAR (X . 9))
          & (BIRD (Y . 9))) -> (SMALLER (X . 9) (Y . 9)))))"

"|wff-10|    ...names...
     ((SUNIV (X . 10)) ((SUNIV (Y . 10)) (((SNAIL (X . 10))
        & (BIRD (Y . 10))) -> (SMALLER (X . 10) (Y . 10)))))"
```

```
"|wff-12|   ...names...
       (($UNIV (X . 12)) (($UNIV (Y . 12)) (((FOX (X . 12))
         & (WOLF (Y . 12))) -> (SMALLER (X . 12) (Y . 12)))))"
"|wff-13|   ...names...
       (($UNIV (X . 13)) (($UNIV (Y . 13)) (((WOLF (X . 13))
         & (FOX (Y . 13))) -> (~ (EATS (X . 13) (Y . 13))))))"
"|wff-14|   ...names...
      (($UNIV (X . 14)) (($UNIV (Y . 14)) (((WOLF (X . 14))
       & (GRAIN (Y . 14))) -> (~ (EATS (X . 14) (Y . 14))))))"
"|wff-15|   ...names...
       (($UNIV (X . 15)) (($UNIV (Y . 15)) (((BIRD (X . 15))
     & (CATERPILLAR (Y . 15))) -> (EATS (X . 15) (Y . 15)))))"
"|wff-16|   ...names...
      (($UNIV (X . 16)) (($UNIV (Y . 16)) (((BIRD (X . 16))
      & (SNAIL (Y . 16))) -> (~ (EATS (X . 16) (Y . 16))))))"
"|wff-17|   ...names...
        (($UNIV (X . 17)) ((CATERPILLAR (X . 17))
      -> (($EXIST (Y . 17)) ((PLANT (Y . 17)) &
            (EATS (X . 17) (Y . 17)))))))"
"|wff-18|   ...names...
       (($UNIV (X . 18)) ((SNAIL (X . 18))
       -> (($EXIST (Y . 18)) ((PLANT (Y . 18)) &
       (EATS (X . 18) (Y . 18))))))"
"|wff-19|   ...names...
   (~ (($EXIST (X . 19)) (SMALLER (X . 19) ($FUN MAGGIE)))))"
"|wff-20|   ...names...
       (~ (($UNIV (X . 20)) (EATS (X . 20) (X . 20))))"
"|wff-21|   ...names...
      ((($UNIV (X . 21))
        ((ANIMAL (X . 21)) -> (~ (PLANT (X . 21)))) &
                  (($UNIV (X . 22)) ((~ (PLANT (X . 22))) ->
        (ANIMAL (X . 22))))))"

"|wff-22|   ...names...
        (~ (($EXIST (X . 23))
        (($EXIST (Y . 23)) ((ANIMAL (X . 23)) &
           ((ANIMAL (Y . 23)) & ((EATS (X . 23) (Y . 23))
           & (($UNIV (Z . 23)) ((GRAIN (Z . 23)) ->
          (EATS (Y . 23) (Z . 23))))))))))"


 The names of two wffs which are to be tried for
 possible resolution are
 |wff-3|
 |wff-7|

THE NEXT RESOLVENT IS GIVEN BY:
"|res-24|   ...names...  (ANIMAL (($FUN TWEEDY)))"
with names of parent-wffs  as ... |wff-3|   and   |wff-7|"

 The names of two wffs which are to be tried for
 possible resolution are
 |wff-4|
 |wff-7|
THE NEXT RESOLVENT IS GIVEN BY:
"|res-25|   ...names...  (ANIMAL (($FUN MAGGIE)))"
  with names of parent-wffs  as ... |wff-4|   and   |wff-7|"
```

```
 THE NEXT RESOLVENT IS GIVEN BY:
 "|res-26|   ...names...  (EATS (($FUN TWEEDY)) (($FUN MAGGIE)))"
  with names of parent-wffs  as ... |wff-15|   and   |wff-7|"

  The names of two wffs which are to be tried for
  possible resolution are
  |wff-22|
  |res-24|
 THE NEXT RESOLVENT IS GIVEN BY:
 "|res-27|   ...names...  (~ (($EXIST (Y . 31)) ((ANIMAL (Y . 31))
                          ((EATS (($FUN TWEEDY)) (Y . 31)) &
(($UNIV (Z . 23) NIL) ((GRAIN (Z . 23)) ->
 (EATS (Y . 31) (Z . 23)))))))))"
with names of parent-wffs  as ... |wff-22|   and   |res-24|"

                        WITH   CHOICE    (ANIMAL (X . 31))
<<<<--   <<<<<---------<<<<<-----------------

  The names of two wffs which are to be tried for
  possible resolution are
  |res-25|
  |res-27|
 THE NEXT RESOLVENT IS GIVEN BY:
 "|res-28|   ...names...
     (~ ((EATS (($FUN TWEEDY)) (($FUN MAGGIE))) &
               (($UNIV (Z . 23) NIL) ((GRAIN (Z . 23)) ->
                          (EATS (($FUN MAGGIE)) (Z . 23)))))))"
  with names of parent-wffs  as ... |res-25|   and   |res-27|"

  The names of two wffs which are to be tried for
  possible resolution are
  |res-28|
  |res-26|
 THE NEXT RESOLVENT IS GIVEN BY:
 "|res-29|   ...names...  (~ (($UNIV (Z . 23) NIL)
  ((GRAIN (Z . 23)) ->  (EATS (($FUN MAGGIE)) (Z . 23)))))).
with names of parent-wffs  as ... |res-28|   and   |res-26|"

                        WITH    CHOICE     (EATS  TWIDY MAGGY)
<<<<<---------<<<<<-----------

  The names of two wffs which are to be tried for
  possible resolution are
  |wff-8|
  |res-25|
 THE NEXT RESOLVENT IS GIVEN BY:
 "|res 30|  ...names...  (($UNIV (X . 37)) (((PLANT (X . 37)) ->
                          (EATS (($FUN MAGGIE)) (X . 37))) V
                        (($UNIV (Y . 37)) (((ANIMAL (Y . 37)) &
                         (SMALLER (Y . 37) (($FUN MAGGIE)))) &
                      (($EXIST (Z . 8) NIL) (((PLANT (Z . 8)) &
          (EATS (Y . 37) (Z . 8))) ->
          (EATS (($FUN MAGGIE)) (Y . 37)))))))))"
      with names of parent-wffs  as ... |wff-8|   and|res-25|

  The names of two wffs which are to be tried for
  possible resolution are
  |wff-6|
```

```
((~ (GRAIN (X . 40))) V ((EATS (($FUN MAGGIE)) (X . 40)) V
                  (($UNIV (Y . 40)) (((ANIMAL (Y . 40)) &
                 (SMALLER (Y . 40) (($FUN MAGGIE)))) &
                (($EXIST (Z . 8) NIL) (((PLANT (Z . 8)) &
     (EATS (Y . 40) (Z . 8))) ->
     (EATS (($FUN MAGGIE)) (Y . 40)))))))))"
 with names of parent-wffs  as ... !wff-6!   and   !res-30!"

     WITH  CHOICE          (PLANT ( X .40))
              <<<<<<<<<<--------<<<-------
```

The names of two wffs which are to be tried for possible resolution are
!res-29!
!res-31!
THE NEXT RESOLVENT IS GIVEN BY:

```
"!res-32!   ...names...
     (($EXIST (Z . 23)) ((EATS (($FUN MAGGIE)) (Z . 23)) V
      (($UNIV (Y . 42)) (((ANIMAL (Y . 42)) &
       (SMALLER (Y . 42) (($FUN MAGGIE)))) &
                        (($EXIST (Z . 8) NIL) (((PLANT (Z . 8)) &
     (EATS (Y . 42) (Z . 8))) ->
       (EATS (($FUN MAGGIE)) (Y . 42)))))))))"
  with names of parent-wffs  as ... !res-29!   and   !res-31!"
```

The names of two wffs which are to be tried for possible resolution are
!res-32!
!res-29!
THE NEXT RESOLVENT IS GIVEN BY:

```
"!res-33!   ...names...
     (($UNIV (Y . 43)) (((ANIMAL (Y . 43))
       & (SMALLER (Y . 43) (($FUN MAGGIE)))) &
            (($EXIST (Z . 8) NIL) (((PLANT (Z . 8)) &
     (EATS (Y . 43) (Z . 8))) ->
      (EATS (($FUN MAGGIE)) (Y . 43))))))"
  with names of parent-wffs  as ... !res-32!   and   !res-29!"

                    WITH   CHOICE   (EATS  MAGGI (Z .23))
  <<<:-- ----<<<<- ------<<<-----
```

The names of two wffs which are to be tried for possible resolution are
!wff-19!
!res-33!
THE NEXT RESOLVENT IS GIVEN BY:

```
  "!res-34!   ...names...  FALSE"
with names of parent-wffs  as ... !wff-19!   and   !res-33!"
```

"The given theorem is valid"

"The set of given wffs alongwith the seq of resolvents is :"

OUR METHOD USES ONLY 11 RESOLUTION STEPS WITHOUT USING ANY STRATEGY, WHEREAS CLAUSAL RESOLUTION USING UNIT RESOLUTION

REFERENCES

[Bha 88] Bhatta, K. S. H. S. R. *Many-Sorted Resolution for Well-formed Formulae with Equality*, M.Tech Thesis, I.I.T, Kanpur.

[BoM 79] Boyer, R. S, and Moore, J.S. *A Computational Logic*, Academic Press, Orlando, Fla., 1979.

[ChL 79] Chang, C.L. and Lee, R.C. *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, Orlando, Fla., 1973.

[DaP 60] Davis, M., and Putnam, H. "A Computing Procedure for *Quantification Theory*",

[Dig 83] Digricoli, V.J. *Resolution by Unification and Equality*, Doctoral Dissertation, New York University, New York, 1983.

[DiH 86] Digricoli, V.J., and Harrison, M.C. "Equality-Based Binary Resolution", *JACM* 33(2), 1986, pp. 253-289.

[Gen 34] Gentzen, G. "Untersuchungen über das Logishe Schliessen", *Mathematishe Zeitschrift* 39, 1934, pp. 176-210, 405-431.

[Hai 56] Hailperin, T. "A Theory of Restricted Quantification I", *JSL* 21(1), 1956.

[Hay 71] Hays, P. "A logic of Actions", *Machine Intelligence* 6, 1971, pp. 495-520.

[Hei 67] Heijenoort, J.V. (editor) *From Frege to Gödel*, Harvard University Press, 1967.

[Hen 72] Henchen, L.J. "N-sorted Logic for Automatic Theorem Proving in

Higher-Order Logic", *Proc. ACM Conference*, Boston, 1972.

[Her 30] Herbrand, J. *Recherches sur la Théorie de la Démonstration*, Dissertation, Univ. of Paris, Warsa, 1930. (Translated as: Investigations in Proof Theory, in *Jacqes Herbrand - Logical writings* (Warren D. Goldfarb, ed.), Harvard Univ. Press, 1971.)

[Jas 34] Jaskowski, S. "On the Rules of Suppositions in Formal Logic", *Studia* 1979. *Logica*, No. 1, Warsa, 1934.

[Kow 79] Kowalski, R. *Logic for Problem Solving*, North Holland Inc., New York,

[Lov 78] Loveland, D.W. *Automated Theorem Proving: A Logical Basis*, North Holland Inc., New York, 1978.

[MaW 80] Manna, Z., and Waldinger, R. "A Deductive Approach to Program Synthesis", *TOPLAS* 2(1), 1980, *pp.* 90-121.

[MaW 86] Manna, Z., and Waldinger, R. "Special Relations in Automated Deduction", *JACM* 33(1), 1986, pp. 1-59.

[Mor 69] Morris, J.B. "E-resolution: An extension of Resolution to include the equality relation", *Proc. of IJCAI*, Washington D.C., 1969, pp. 287-294.

[Mur 82] Murray, N.V. "Completely Non-Clausal Theorem Proving", *AI* 18(1), 1982, *pp.* 67-85.

[Pel 86] Pelletier, F.J. "Seventy-Five Problems for testing Automatic Theorem Provers", *J. of Automated Reasoning* 2, 1986, *pp.* 191-216.

[Qui 61] Quine, W.V. *Methods of Logic*, Holt, Reinhart and Winston Inc., New York, 1961.

[Rob 65] Robinson, J.A. "A Machine-Oriented Logic Based on Resolution

Principle", *JACM* 12(1), 1965, pp. 23-41.

[San 87] Sangle, R. *Programming Paradigms in LISP*, Department of Computer Science and Engineering, I.I.T, Kanpur.

[SFGMW 84] Steele, G.L., Fahlman, S.E., Gabriel, R.P., Moon, D.A., and Weinreb, D.L. *Common Lisp*, Digital Press, 1984.

[Sch 38] Schmidt, A. "Über deductive Theoien mit Mehreren Sorten von Grunddigen", *Mathematische Annalen* 115, 1938, pp. 485-506.

[Sch 85] Schmidt-Schauss, M. *Mechanical Generation of Sorts in Clause Sets*, SEKI Memo. Univ. of Keiserslautern, WG, 1985.

[Sha 79] Shapiro, Stuart C., *Techniques of Artificial Intelligence* D. Van Nostrand Company

[Sko 20] Skolem, Thorlf, "Logisch-Kambinatorische untersuchungen über die Erfüllbarkeit oder Beweisbarkeit mathematischer Sätze nebst einem Theorem über dichte mengen", *Videnskopsselskapits skifter I. Mathematik-naturvidenskabelig klasse*, No. 4, 1920.

(Translated as: Logico-Combinatorial investigations in the satisfiability or provability of mathematical propositions: A simplified proof of a theorem by L. Löwenheim and generalizations of the theorem [Hei 67]

[Sko 28] Skolem, Thoralf, " über die mathematische Logik", *Norsk matematisk tideskrift* **10**, 1928, pp. 125-142. (Translated as: On Mathematical Logic [Hei 67])

[Sti 85] Stickel, M.E. "A Non-Clausal Connection-Graph Resolution Theorem Proving Program", *Proc. Nat. Conf. on AI*, Pittsberg, 1985 and *American Association for AI*, Menlo Park, pp. 229-233.

[Tar 51] Tarski, Alfred *A Decision Method for Elementary Algebra and Geometry*, Berkley, 1951.

[Wal 82] Walther, Christoph *A many-Sorted Calculus based on Resolution and Paramodulation*, SEKI Memo. Univ. Karlsruhe, WG, 1982.

[Wan 52] Wang, H. "Logic of Many-Sorted Theories", *JSL* 17(2), 1952.

[Wan 60] Wang, H. "Toward mechanical mathematics", *IBM Journal of Reseach and Development*, 4: 2-22. Reprinted in *The Modelling of Mind: Computers and Intelligence* (Sayre, K. M.and Crossman, F. j., eds.), New York: Simon and Schuster, 1963.

[WOL 84] Wos, L., Overbeek, R. Lusk, E., and Boyle, *J. Automated Reasoning - Introduction and Applications*, PHI, NJ, 1984.

[WoR 69] Wos, L.A., and Robinson, G.A., "paramodulation and Theorem Proving in First-Order Theories with Equality", *Machine Intelligence* 4, 1969, pp. 135-150.